

2001

A Model for Managing Information Flow on the World Wide Web

Evans, Michael Paul

<http://hdl.handle.net/10026.1/869>

<http://dx.doi.org/10.24382/1365>

University of Plymouth

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

Copyright Statement

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognize that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.

A Model for Managing Information Flow on the World Wide Web

by

MICHAEL PAUL EVANS

A thesis submitted to the University of Plymouth in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

Department of Communications and Electrical Engineering
Faculty of Technology

March 2001

Michael Paul Evans

A Model for Managing Information Flow on the World Wide Web

Abstract

This thesis considers the nature of information management on the World Wide Web. The web has evolved into a global information system that is completely unregulated, permitting anyone to publish whatever information they wish. However, this information is almost entirely unmanaged, which, together with the enormous number of users who access it, places enormous strain on the web's architecture. This has led to the exposure of inherent flaws, which reduce its effectiveness as an information system.

The thesis presents a thorough analysis of the state of this architecture, and identifies three flaws that could render the web unusable: link rot; a shrinking namespace; and the inevitable increase of noise in the system. A critical examination of existing solutions to these flaws is provided, together with a discussion on why the solutions have not been deployed or adopted. The thesis determines that they have failed to take into account the nature of the information flow between information provider and consumer, or the open philosophy of the web. The overall aim of the research has therefore been to design a new solution to these flaws in the web, based on a greater understanding of the nature of the information that flows upon it.

The realization of this objective has included the development of a new model for managing information flow on the web, which is used to develop a solution to the flaws. The solution comprises three new additions to the web's architecture: a temporal referencing scheme; an Oracle Server Network for more effective web browsing; and a Resource Locator Service, which provides automatic transparent resource migration. The thesis describes their design and operation, and presents the concept of the Request Router, which provides a new way of integrating such distributed systems into the web's existing architecture without breaking it. The design of the Resource Locator Service, including the development of new protocols for resource migration, is covered in great detail, and a prototype system that has been developed to prove the effectiveness of the design is presented. The design is further validated by comprehensive performance measurements of the prototype, which show that it will scale to manage a web whose size is orders of magnitude greater than it is today.

Table of Contents

ABSTRACT	I
TABLE OF CONTENTS	II
LIST OF FIGURES	IX
LIST OF TABLES	XI
ACKNOWLEDGEMENTS	XII
AUTHOR'S DECLARATION	XIV
1. INTRODUCTION AND OVERVIEW	1
1.1 INTRODUCTION.....	2
1.2 AIMS AND OBJECTIVES.....	4
1.3 THESIS STRUCTURE.....	7
2. THE WORLD WIDE WEB.....	10
2.1 INTRODUCTION.....	11
2.2 THE ARCHITECTURE OF THE WORLD WIDE WEB	13
2.2.1 <i>The Web's Origins</i>	13
2.2.2 <i>Architectural Overview</i>	14
2.2.2.1 HyperText Transfer Protocol.....	16
2.2.2.2 The Uniform Resource Identifier.....	17
2.2.2.3 HyperText Markup Language.....	18
2.2.3 <i>The Relationship Between the Web and the Internet</i>	19
2.2.4 <i>A Philosophy of No Control</i>	21
2.3 MANAGING THE WEB'S INFORMATION.....	22

2.3.1	<i>Identifying Information</i>	23
2.3.2	<i>Structuring Information</i>	26
2.3.3	<i>Retrieving Information</i>	28
2.3.3.1	Overview of the Search Engine.....	29
2.4	THE INFORMATION MANAGEMENT DICHOTOMY	32
2.4.1	<i>Gatekeeping</i>	34
2.4.2	<i>Case Study: Proprietary Online Information Providers</i>	35
2.4.3	<i>Internet Anarchy</i>	39
2.5	SUMMARY	42
3.	FLAWS IN THE WEB'S ARCHITECTURE	44
3.1	INTRODUCTION.....	45
3.2	LINK ROT	48
3.2.1	<i>The Cause of Link Rot</i>	49
3.2.2	<i>The Damaging Effects of Link Rot</i>	52
3.2.3	<i>Measuring Link Rot in the web</i>	54
3.2.3.1	Link Rot Incidence	55
3.2.3.2	Link Rot Prevalence	55
3.2.3.3	The Life Span of a Web Resource.....	56
3.2.3.4	An Attempted Experiment to Determine the True Level of Link Rot in the Web.....	58
3.2.3.5	Determining Link Rot from the Literature.....	59
3.2.4	<i>Existing Solutions to Link Rot</i>	60
3.2.4.1	Resource Migration Mechanisms.....	60
3.2.5	<i>Summary of the Link Rot Problem</i>	74
3.3	SHRINKING NAMESPACE	74
3.3.1	<i>The Cause of the Shrinking Namespace</i>	75
3.3.2	<i>The Damaging Effects of the Shrinking Namespace</i>	76
3.3.3	<i>Determining the Extent of the Problem</i>	78
3.3.4	<i>Solutions to the Shrinking Namespace Problem</i>	81

3.3.4.1	The Irreplaceable DNS.....	82
3.3.4.2	The Inextensible DNS.....	87
3.3.5	<i>Summary of the Shrinking Namespace Problem</i>	90
3.4	INCREASING NOISE.....	90
3.4.1	<i>The Cause of the Increasing Noise</i>	91
3.4.2	<i>The Damaging Effects of Noise</i>	92
3.4.3	<i>Determining the Extent of the Problem</i>	93
3.4.3.1	The State of Hyperlink Navigation.....	94
3.4.3.1.1	Navigation Mechanisms.....	94
3.4.3.1.2	The Problem with Browsing.....	96
3.4.3.1.3	The State of the Web's Hyperlink Structure.....	97
3.4.3.2	The State of the Web's Information Retrieval Services.....	99
3.4.3.2.1	Coverage-Oriented Services.....	100
3.4.3.2.2	Relevance-Oriented Services.....	103
3.4.3.3	Implicit Gatekeeping.....	104
3.4.4	<i>Summary of the Increasing Noise Problem</i>	105
3.5	SUMMARY.....	106
4.	HOMINID – A MODEL FOR MANAGING INFORMATION FLOW ON THE WEB.....	108
4.1	INTRODUCTION.....	109
4.2	THE CORE COMPONENTS OF THE HOMINID MODEL.....	110
4.3	REDUCING LINK ROT.....	110
4.3.1	<i>Managing Content Migration with Temporal References</i>	113
4.3.2	<i>Managing Resource Migration with the Resource Locator Service</i>	116
4.4	EASING THE NAMESPACE PRESSURE.....	118
4.4.1	<i>Shrinking Namespace Increases Pressure</i>	119
4.4.1.1	Exploitative Strategies.....	120
4.4.1.2	The Problem With ICANN's Solution.....	121
4.4.2	<i>Easing the Pressure</i>	123

4.4.3	<i>Defining the Semantics of the New Namespace</i>	123
4.5	REDUCING THE NOISE IN THE WEB.....	125
4.5.1	<i>The Deceptive Hyperlink Versus the User</i>	126
4.5.1.1	Deception as an Effective Strategy.....	127
4.5.1.2	How the Hyperlink Breaks the Flow of Information.....	129
4.5.2	<i>The Deceptive Web Site Versus the Search Engine</i>	131
4.5.2.1	An Arms Race Between the Search Engine and the Web Resource.....	132
4.5.2.2	The Fight For Relevance.....	133
4.5.3	<i>A Persistent Problem</i>	136
4.5.4	<i>The Oracle Server – A Novel Platform for Enhanced Navigation</i>	136
4.5.4.1	Resolving the Information Management Dichotomy.....	138
4.5.4.2	Functional Operation of the Oracle Server.....	138
4.5.4.2.1	Characteristic Infons.....	139
4.5.4.2.2	Navigational Infons.....	140
4.5.4.2.3	The Heuristics of the Oracle Server	142
4.5.4.3	New Web Metrics	143
4.6	SUMMARY	144
5.	ARCHITECTURAL DESIGN OF THE HOMINID MODEL	148
5.1	INTRODUCTION.....	149
5.2	DESIGNING THE RESOURCE LOCATOR SERVICE	149
5.2.1	<i>The Scope of the Resource Locator Service</i>	149
5.2.2	<i>Selecting the Approach to Resource Migration</i>	150
5.2.3	<i>Removing the Namespace Constraints</i>	151
5.2.4	<i>Defining the Locator's Client-Side Interface</i>	153
5.2.5	<i>Missing Mediation</i>	155
5.3	REQUEST ROUTING: NOVEL MEDIATION BETWEEN THE WEB AND A DISTRIBUTED SYSTEM.....	156
5.3.1	<i>The CARP Hash Routing Algorithm</i>	157
5.3.2	<i>How the CARP Hash Routing Algorithm works</i>	158

5.3.3	<i>Adapting the CARP Hash Routing Algorithm for the RLS</i>	159
5.3.3.1	Updating the Request Router	161
5.3.3.2	Backwards Compatibility	163
5.3.4	<i>How the Hash Routing Algorithm Works in the RLS</i>	165
5.3.5	<i>The Design of the Request Router</i>	168
5.3.5.1	The Request Router's Interfaces	170
5.3.6	<i>Scalability</i>	172
5.3.6.1	Network Overhead	172
5.3.6.2	CPU Overhead	173
5.3.6.3	Scalability of the Overall Design	173
5.3.7	<i>Resilience</i>	174
5.3.8	<i>Impact of the Resource Locator Service on existing Web mechanisms</i>	175
5.3.8.1	Impact on Caching Servers	175
5.3.8.2	Impact on History and Bookmark Mechanisms	177
5.4	TEMPORAL REFERENCES	177
5.4.1	<i>The URL Extension</i>	178
5.4.2	<i>The Temporal URL Scheme</i>	179
5.4.3	<i>Defining the Scope of the Temporal Reference</i>	181
5.4.3.1	The URL Extension Versus the Temporal URL	182
5.5	DESIGNING THE ORACLE SERVER	183
5.5.1	<i>The Oracle Server Network</i>	183
5.5.2	<i>The Architecture of the Oracle Server Network</i>	184
5.5.3	<i>Obtaining the Infons</i>	185
5.5.3.1	Navigational Infons	186
5.5.3.2	Characteristic Infons	187
5.5.4	<i>The OSN as a Platform for New Services</i>	188
5.6	SUMMARY	189
6.	THE RESOURCE LOCATOR SERVICE	192

6.1	INTRODUCTION.....	193
6.1.1	<i>Protocol Development</i>	193
6.2	MIGRATING RESOURCES WITH THE RESOURCE MIGRATION PROTOCOL.....	195
6.2.1	<i>Applying the WebDAV Protocols to Resource Migration</i>	196
6.2.1.1	Security.....	196
6.2.1.1.1	Application to Resource Migration.....	197
6.2.1.2	Safe File Transfer.....	197
6.2.1.2.1	Application to Resource Migration.....	198
6.2.1.3	Server Querying.....	198
6.2.1.3.1	Application to Resource Migration.....	199
6.2.2	<i>Disadvantages of Using WebDAV for Resource Migration</i>	199
6.2.3	<i>The Specification of the Resource Migration Protocol</i>	200
6.2.3.1	The Migration Process.....	201
6.2.3.2	Access Control and Authorization.....	204
6.2.3.3	Safe File Transfer.....	205
6.2.3.4	Updating the Locator.....	207
6.2.3.5	Resource Replication.....	207
6.2.3.6	Resource Migration using Non-WebDAV Compliant Servers.....	209
6.3	RECONFIGURING THE RLS VIA THE LOCATOR CONTROL PROTOCOL	210
6.3.1	<i>The Record Migration Process</i>	211
6.3.2	<i>Managing the Addition of a New Locator</i>	212
6.3.2.1	Overview	212
6.3.2.2	Message Sequence Chart for Adding a New Locator.....	213
6.3.3	<i>Managing the Removal of an Existing Locator</i>	221
6.3.3.1	Overview	221
6.3.3.2	Message Sequence Chart for Removing an Existing Locator.....	224
6.3.4	<i>Performance Implications of the LCP</i>	228
6.4	A PROTOTYPE RESOURCE LOCATOR SERVICE.....	231
6.4.1	<i>A Prototype Locator</i>	233

6.4.2	<i>A Prototype Request Router</i>	235
6.4.3	<i>A Prototype Management Interface</i>	236
6.4.4	<i>Implementing the Resource Migration Protocol</i>	237
6.4.5	<i>Performance</i>	238
6.4.5.1	Network Overhead.....	238
6.4.5.2	CPU Overhead	238
6.4.5.3	Total System Overhead	240
6.4.5.4	The Cost of Changing the Configuration.....	242
6.4.5.5	Performance Summary.....	249
6.4.6	<i>Demonstrating New Services with the Prototype RLS</i>	249
6.4.6.1	Load Balancing.....	250
6.4.6.2	Fault Tolerance	251
6.4.6.3	Mobile Agents.....	252
6.4.6.4	Other Enhanced Services	255
6.5	SUMMARY	256
7.	CONCLUSION	257
7.1	ACHIEVEMENTS OF THE RESEARCH PROGRAMME	258
7.2	LIMITATIONS OF THE RESEARCH	259
7.3	SUGGESTIONS AND SCOPE FOR FUTURE WORK	260
7.4	THE FUTURE OF THE WORLD WIDE WEB	263
	LIST OF REFERENCES	267
	APPENDIX A	290
	APPENDIX B	292
	APPENDIX C	294
	APPENDIX D	295

List of Figures

Figure 1 - Screenshot of the Google Search Engine.....	30
Figure 2 - Total number of domain names registered by quarter (DotCom, 2000).....	79
Figure 3 - Percentage of registered domain names according to TLD (DotCom, 2000).....	80
Figure 4 - Hyperlink structure of the web (Broder et al., 2000).....	99
Figure 5 - Relevance of the Web Compared to Dialog (Feldman, 1998).....	102
Figure 6 - The Result of Content Changing Within a Resource.....	112
Figure 7 - Temporal Referencing.....	115
Figure 8 - Real Example of Fake User Interface	127
Figure 9 - Fake User Interface Imitators.....	128
Figure 10 - The Browser's Status Bar as a Navigation Aid.....	129
Figure 11 - Number of Queries per Day for the Popular Search Engines (Sullivan, 2000c).	132
Figure 12 - A High-Level Overview of the RLS	154
Figure 13 - How the RR updates itself.....	162
Figure 14 - The Architecture of the Resource Locator Service.....	166
Figure 15 - Sample JavaScript function showing a RR embedded in a HTML Page.....	170
Figure 16 - The Architecture of the Oracle Server Network.....	185
Figure 17 - The Navigational Infos passed from the client to the OSN via the RLS.....	186
Figure 18 - MSC for Resource Migration Protocol (assuming successful migration).....	203
Figure 19a-c - Managing the Addition of a New Locator.....	213
Figure 20 - MSC Describing the Locator Addition Process in the LCP.....	215

Figure 21 - Example PUTREC message.....	219
Figure 22a-c - Managing the Removal of an Existing Locator with the LCP.....	222
Figure 23 - MSC for Removing a Locator.....	223
Figure 24 - Architectural Design of the Prototype RLS.....	233
Figure 25 - RLS Management Interface	237
Figure 26 - Performance Results of the Prototype Request Router.....	239
Figure 27 - Number of Messages Sent According to Configuration of RLS and Number of Resources Managed	243
Figure 28 - Total time taken to add a new Locator according to Locator number.....	248
Figure 29 - Prototype Fault Tolerance Application of the RLS.....	252
Figure 30 - Prototype Mobile Agent Demonstration.....	254

List of Tables

Table 1 - Analysis of Existing Migration Mechanisms	64
Table 2 - Name Systems in Use Today.....	83
Table 3 - New TLDs submitted to ICANN.....	88
Table 4 - New TLDs chosen by ICANN.....	88
Table 5 - The Core Components of the HOMINID Model.....	146
Table 6 - Results of the Overhead Introduced by the RLS.....	241

Acknowledgements

This study was undertaken in part to further my own knowledge, and in part to further the knowledge of the web community in order that the web remains society's principal information system. Much of the research could not have been conducted without the open philosophy of the web, which encourages research to be shared, and its rich interconnected nature, which makes for easy access to that research. It is hoped after the conclusion of this study, that access can be made even easier.

,

The European ACTS project DOLMEN provided both the principal funding for the first two and a half years of study, and the foundation for the idea of the research. DOLMEN highlighted the need for proper information management in an information system, but its methods, coming from a telecommunications perspective, seemed so at odds with the web's open philosophy, that the need for a new model of management seemed obvious and imperative.

The work could not have been done without the guidance of my Director of Studies, Dr. Steven Furnell, who helped me through the DOLMEN project, guided me through the process of publishing and presenting papers, worked tirelessly to secure most of my further funding, and who provided the model for the shape of this thesis; the reason for its size and comprehensive nature can therefore be firmly laid at his door.

I would also like to thank my supervisors, Prof. Peter Sanders and Prof. Paul Reynolds, both of whom provided me with the encouragement and technical advice that was much needed along the way. I would particularly like to thank Prof. Reynolds for his part in securing funding from Orange Personal Telecommunications in the form of two research projects, without which the work could not have been completed. Thanks are also owed to my colleague, Paul Dowland, who was put upon at the last minute to help with some of the experiments conducted as part of this work in spite of his own pressing schedule.

I would also like to thank my friends and family for the support that they gave. Despite not seeing me for much of the writing-up stage, they continued to encourage my efforts, and not taken my absence personally.

Finally, the biggest thanks of all belongs to my fiancée, Vicky, who has had to put up with minimal income, expanding deadlines, stress, and little support from me during a period of change and insecurity for us both. Throughout it all, she has remained wholly supportive and understanding, despite the fact that this thesis was written while I worked full-time. As such, the thesis has dominated my time at the expense of the time I have for her; it is time now that I put that right.

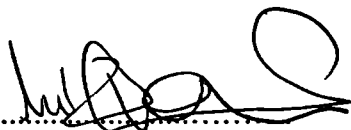
Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was financed with primary funding from the EU ACTS DOLMEN project.

Relevant conferences and DOLMEN project meetings were regularly attended (at which work was frequently presented) and a number of external establishments were visited for consultation purposes. In addition, several papers were prepared for publication, details of which are listed in the appendices.

Signed



Date

9.9.01

1. Introduction and Overview

The research presents a new model for managing information flow on the web that has been designed according to the web's open philosophy. The model solves the problems of link rot, a shrinking namespace, and increasing noise in the system, and has been designed to integrate into the web's existing architecture without breaking it.

1.1 Introduction

The World Wide Web (web) has transformed society since its inception in 1989, becoming the most important sociological invention of the late 20th Century. Many other network applications existed prior to the web, and countless others have been developed since, but none have had such an impact on society. The web is unique in that it provides a common information space that is global in scope, but which does not discriminate according to information or users. Effectively, anyone can publish anything on the web, enabling it to evolve into an abstract representation of society's ideas, thoughts, opinions, fashions, and fears. It is the combination of the web's lack of censorship, its openness, its global reach, and the ease with which information can be published that has given the web such impact on society. For the first time in history, people with limited technological knowledge can publish whatever information they wish and have it seen across the world instantly.

The web has impacted on many different facets of life: Its explosive growth has remained exponential since its birth; fortunes have been made and lost; governments have tried to curtail its freedom; business has been transformed through e-commerce; and security and privacy issues have been ever-present. All these issues are symptomatic of the fact that the web is not just a technological innovation, but a societal innovation as well, affecting every level of people's lives, either directly or indirectly.

The reason for the web's impact lies in its open design and philosophy. Specifically, all of the web's protocols and standards are freely available, allowing anybody to read and modify them

if required, while its federated design, inherited in part from its reliance on the Internet, ensures that no central organization controls it. Thus, the web is open from an architectural level, and from a user level, both of which give it a competitive advantage compared to similar information systems, such as America Online, which have a centralized controlling organization behind them.

However, the web is not without its problems. Without any controlling organization, the web has no form of censorship, and so information of low quality competes directly with information of high quality on an equal level, with little to distinguish between the two in the eyes of the user. Worse, information can be deliberately misleading, designed to confuse the user for the purposes of the information provider. The end result is an information explosion, most of it of dubious quality, which ultimately degrades the quality of the service that the web provides. In addition, the architecture of the web was never designed to scale to the size that it has, and each new user and new item of information adds to the pressure placed upon its foundations. As this thesis aims to show, cracks are appearing in the web as flaws that are inherent in its architecture begin to undermine its structure and diminish its role in society.

Many solutions exist that attempt to solve these problems, but their effectiveness has been limited at best. They have often been isolated systems that attempt to patch a specific weakness without taking into account problems in other areas of the web's architecture. Many of these solutions also require the web's architecture to be adapted, which ultimately would force all existing web entities to be redesigned to take advantage of the solution offered.

However, in a system the size of the web, this is impractical at best, and, as the low adoption rates of these solutions has proved, impossible in practice.

The aim of the research presented in this thesis, therefore, is to design an effective solution to the most critical flaws of the web without requiring existing entities to be redesigned. However, rather than basing the solution on an existing model of information flow in a network, the research takes the perspective of information flowing from information provider to information consumer, effectively ignoring its flow through the technological foundations of the web altogether. Using this perspective, a new model for managing information flow in the web has been designed, which works with the web and its information, rather than against it. The model manages the web's information without censoring it, and solves some of the web's flaws without requiring the modification of its architecture.

1.2 Aims and Objectives

This study is concerned with the architecture of the web and the information published on it. More specifically, it identifies the problems in the web's information space and focuses on the architectural weaknesses that have brought about these problems, with the aim being to solve the problems in a way that is sympathetic to the underlying philosophy of the web's design principles.

In order to do this, the research programme can be divided into three key phases. The first phase provides a comprehensive analysis of the web's architecture in its current state, and focuses on the three areas of weakness that provide the greatest cause for concern: identifying,

structuring, and retrieving information. Specific problems are identified within these areas that could potentially halt the growth of the web altogether. The second phase of the programme aims to solve these problems through the development of a new model for managing the information flow on the web without breaking its architecture or damaging its culture. The third and final phase involves the complete specification and development of part of the model. In order to validate the design, this stage also includes the development and testing of a prototype, with performance measurements provided to demonstrate its effectiveness in a real world setting.

A principal objective of the first phase was to identify those flaws in the web's existing architecture that were deemed most damaging to its future. In order to do this, an extensive literature search was performed in order to determine the nature and extent of the flaws, while an experiment was conducted in the hope of gaining empirical data on the extent of the problems identified. Once the problems had been identified, they remained as the focus of the remainder of the research programme.

The objective of the second phase was to develop a new model for managing information flow on the web, which represents the focus of the research. The model comprises a new mechanism for enhanced navigation across the web; a new addressing scheme that can reference resources according to time and space; and a replacement for the DNS that has can transparently migrate resources across web servers. The model relies upon a new form of mediation, called request routing, which enables these new distributed systems to be integrated into the web without breaking its existing architecture.

Finally, the objective of the third phase was to develop the model such that it could be deployed on the web to provide real solutions to the identified flaws. As such, this phase involved completely specifying the design of the model from an engineering perspective, and developing, testing, and measuring a prototype.

These objectives can be more formally specified as follows:

- To assess the current state of the web's architecture.
- To identify the flaws in the web's architecture specific to identifying, structuring, and retrieving information, and the extent to which they might affect its growth and effectiveness as an information system.
- To develop a new model for managing the web's information flow, and ultimately provide a solution to the identified flaws of the web.
- To design and specify a system that implements the new model.
- To build and test a prototype of this system to demonstrate its effectiveness within the web.

These objectives correspond to the general sequence of the material presented in the subsequent chapters of the thesis, as will be discussed in the next section.

The research has involved significant liaison with networks and telecom engineers (particularly during the early stages of the work). The majority of this consultation occurred within the ACTS DOLMEN project, which was concerned with the development of a service architecture for fixed and mobile networks, based on TINA-C (TINA, 1994). Other consultation occurred within the WebDAV working group, and through discussions with various experts in the field.

1.3 Thesis Structure

This thesis describes the research leading to the formulation of a new model for managing information flow on the web, which is used to provide a solution to key flaws within the web's architecture. The foundations for the thesis are provided in chapter 2, which begins by examining the web's existing architecture in detail, focusing on the way in which it identifies, structures, and retrieves its information, and highlighting the effect that this has on the system as a whole. The chapter reveals the dichotomy that information management introduces to a global information system, with an open design philosophy leading to a popular but ineffective system, and a closed philosophy leading to an unpopular but effective system. This forms the core focus for the research, which aims to significantly improve the web's management of its information without destroying its openness and popularity.

Chapter 3 provides substantive evidence of the flaws in the web's architecture that are a direct result of its open philosophy, and which are steadily eroding its effectiveness as its popularity increases. An exhaustive literature search is presented that reveals the extent of the web's problems, and the existing solutions that have been proposed, but which have ultimately

failed. To provide further detail, the chapter discusses an experiment conducted as part of the research programme, which was designed to provide empirical data on the problem of link rot in the web, but which was prematurely terminated. The chapter effectively underpins the research by identifying the scope of the web's problems, and concludes that a new, human-oriented model for managing information flow is required in order to solve them.

Chapter 4 presents such a model. HOMINID, as the model is called, provides a new way of managing information flow on the web that is designed to work according to the nature of the information that flows upon it. The chapter provides a conceptual overview of the model, which includes the design of a new, web-specific Resource Location Service; the introduction of time as a new dimension to the web; and a system for universal access to meta-data and navigational information to help decrease the level of noise in the web without falling foul of the information management dichotomy defined in chapter 2.

Chapter 5 presents a detailed specification of the HOMINID model, showing how it can be applied to the web. The chapter essentially provides a blueprint for deploying the model on the web, including a functional design specification of its core concepts. The chapter describes the definition of new services, URI schemes, management entities and their interfaces, and routing objects that can be deployed in the web's architecture in a way that is backwards-compatible with existing web entities.

Chapter 6 defines the complete design, specification, and implementation of the HOMINID model's Resource Locator Service, including the full specification of the required protocols,

interfaces and objects that are needed for deployment. A new Resource Migration Protocol is defined, and a prototype of the service is described that has been developed according to this specification, in order to validate its design. The performance of the prototype has been measured, and extensive results are provided in full, in order to demonstrate the effectiveness of the design in today's web.

Finally, chapter 7 presents the main conclusions arising from the entire research programme, highlighting the principal achievements and limitations of the work, along with suggestions for potential further development. The thesis also includes a number of appendices, which contain a variety of additional information in support of the main discussion (including a number of published papers from the research programme).

2. The World Wide Web

The World Wide Web (web) is a global information system that provides equal access to information providers and information users. Its success is due in no small part to its philosophy of openness and extensibility. Anyone can develop anything for the web, and market forces alone will determine its success. This has assured unprecedented investment in the web's technology, accelerating the rate of technological progress, and connecting society and its information. However, it has also left it with no central controlling authority, and no explicit standards for managing its information, leading to an anarchic state of information provision that progressively weakens the web's use as an effective information system.

2.1 Introduction

The late 20th Century saw the emergence of global information systems that could provide information and services on an unprecedented scale. The global communications networks supporting these systems came from the fields of telecommunications and computing, with ideas and technologies crossing the two domains to provide solutions to common problems. These global networks, supported by many users, are different from local networks, as the services they provide have the power to impact on society. This impact becomes more powerful as the services become richer, and more able to provide a diverse range of information.

The telephone system came from the field of telecommunications, and its services originally consisted primarily of transporting a human voice from one telephone to another. Over time, its services have become richer, and now telecommunications companies (telcos) have begun to extend their role by providing multimedia-based services such as graphics and video. However, a new global network emerged from the computing field, in the form of the Internet and, more recently, the World Wide Web (web). The Internet experienced a global surge in popularity, largely through the development of the web and the introduction of the graphical web browser *Mosaic* by NCSA (Berners-Lee and Fischetti, 1999). The web has developed into a competing platform to the telecommunications system, capable of providing the same rich multimedia services that the telecommunications companies wish to provide, but based on a different paradigm than the telecommunications model.

Whereas the telecommunications model tightly integrates the provision of services with the control of the network, the Internet, upon which the web runs, effectively separates the services from the network. This is reflected in the architecture of the two models. The telecommunications model designs all-encompassing architectures that control every part of the network, from the user and their terminal, to the network resources switching data deep within the network, and provides well-managed services in a tightly controlled environment. In contrast, the Internet model focuses more on the reliable transport of data, and leaves others to design whatever service they require. The Internet completely ignores the network resources that support it, and the web provides a platform for service provision without regard for what services are deployed on it. The result is seen either as “Internet anarchy” (Raatikainen et al., 1998) or as an “open, federated system” (Berners-Lee and Fischetti, 1999), depending on your chosen philosophy.

Despite the existence of the two models, it is the Internet and the web that have become pervasive in the developed world, becoming the *de facto* global information system. The reasons behind the web’s success will be examined in section 2.4.1, but before that, the following sections will examine the architecture of the web, and analyse how its information is managed.

2.2 *The Architecture of the World Wide Web*

2.2.1 *The Web's Origins*

The Internet is a data network from the field of computing that has gradually evolved over time to become a single ubiquitous network that enables all other data networks to communicate and interoperate with one another. The 1990s saw an explosion both in the number of people using the Internet, and the types of services it could offer. As the Internet grew in popularity, however, users found they needed a consistent, standard way of formatting and referencing information. Such a system was developed by Tim Berners-Lee, of the European particle physics research laboratory, CERN, who proposed the World Wide Web as a hypertext distribution system that could be accessed by any type of computer across the world, and which would present its information through a single user-interface. Since its invention, the web now has a user population of over 369 million (GlobalReach, 2000), yet it was conceived of by one man, designed by a small team of developers, and refined by many different organizations and companies. This was possible due to the open nature of the Internet. Essentially, anybody can develop a service for deployment on top of the Internet, and not worry about the underlying network. Better still, unlike the situation for telecommunications service providers, service deployment does not require negotiating access to the network, payment to the network operator (apart from a minimal charge to an Internet Service Provider, although even this is now free in the UK), or the development of a new architecture based around the new service.

This dramatically reduces both the cost of new services, and the time from service conception to service deployment.

2.2.2 Architectural Overview

The World Wide Web is defined as:

“...the universe of network-accessible information, the embodiment of human knowledge...The Web has a body of software, and a set of protocols and conventions. Through the use of hypertext and multimedia techniques, the web is easy for anyone to roam, browse, and contribute to.” (W3C, 1992).

As such, the definition of the web strongly reflects the philosophy of the Internet model: ignore the details of the underlying system, manage only the parts you need to manage, and let anyone use your services.

The web is a loosely distributed system based on the client-server model, whereby a client must explicitly request a resource from a server. A resource is defined as “anything that has identity” (Berners-Lee et al., 1998, p2), and usually includes such things as documents or images. These currently form the main items of traffic on the web, and are transferred from server to client using the HTTP protocol (Berners-Lee et al., 1996).

The web’s core architecture comprises three standards:

- HyperText Transfer Protocol (Berners-Lee et al., 1996).
- The Uniform Resource Identifier (Berners-Lee et al., 1998).
- HyperText Markup Language (Berners-Lee et al., 1995).

The HyperText Transfer Protocol (HTTP) is the protocol used to distribute this information over the existing Internet architecture; the Uniform Resource Identifier (URI) is used to both locate and identify resources (text, images, files, etc.), and represents the standard that defines the operation of the hyperlink, enabling one resource to be linked to another; and HyperText Markup Language (HTML) is used to present the information contained within the resources in a consistent way across all computer platforms.

These three standards are implemented in a web browser, which presents the information to the user, and in a web server, which hosts the information and transmits it to the user upon request. The browser also uses other third-party standards for displaying content of a particular media type, such as images, sound, animation, etc.

The user interacts with the web through the web browser, which acts as the web's user interface and renders all content delivered to the user. The user begins the process by typing the URI of a resource (usually a HTML document, which is known as a *web page*) into a browser. The URI acts as the resource's address, uniquely locating the resource on a server. It encodes the server's domain name, and also the directory location and file name of the resource on that server. A web browser, given a resource's URI, can locate the resource and download it from its server. The resource's content is then extracted and rendered in the browser's window using HTML encoding rules. This is the mechanism through which the web provides its

services. The following sub-sections examine these three standards in greater detail, and describe the role each has to play in the architecture of the web.

2.2.2.1 HyperText Transfer Protocol

The web relies on HTTP as its transfer protocol, transferring its resources from server to client. The protocol is text-based and therefore extremely simple when compared with other protocols. All it is concerned with is the transfer of a web resource (such as a web page or an image) across a reliable transport connection. As such, TCP/IP is nearly always used as the transport protocol, and will be considered as the default transport protocol for this thesis. TCP/IP ensures the data is delivered safely, leaving HTTP free to manage application-level issues. These include content negotiation, whereby the server can negotiate with the client as to which version of a resource is best rendered on the client's access device; server redirection, whereby a server redirects the request to another server; and authorization, although this is not highly robust in HTTP.

HTTP treats all resources as if they were files located on a server, with a URI acting as their identifier. Although it is able to differentiate between different content types (such as HTML documents and GIF images) using the Internet Media Types specification (Postel, 1996) and the Multipurpose Internet Mail Extensions (MIME) specification (Freed and Borenstein, 1996), it does not exclude one content type over another, and although it may encode or transform the content, it never alters it.

HTTP is an extensible protocol, enabling it to be adapted and extended through a well-defined extension framework (Nielsen et al., 2000). Recent extensions have included a *Capabilities and Preferences Protocol* (Ohto and Hjelm, 1999), and the new *WebDAV* protocol for distributed authoring and versioning across the web (Goland et al., 1999), each of which has provided new semantics and message headers to HTTP, but without breaking the existing core protocol. The extensibility of HTTP enables it to be used in a wide range of situations, and helps prolong its usefulness in an environment as dynamic as the web.

2.2.2.2 *The Uniform Resource Identifier*

The Uniform Resource Identifier (URI) (Berners-Lee et al., 1998) is a generic syntax that provides a simple and extensible means of identifying a resource. As such, it is used to compose all identifiers on the web, and can be seen as a superset of all web identifiers. The most commonly used type of URI is the *Uniform Resource Locator* (URL) (Berners-Lee et al., 1994), which is the subset of URIs that identify a resource through its location (Berners-Lee et al., 1998). There are many other types of URI scheme, such as that based on the *Handle System* (Sun and Lannom, 2000), or the *Uniform Resource Name* (Berners-Lee et al., 1998), but because of their failure to be widely adopted on the web, the URL is perceived as the only type of identifier. Despite this, the distinction between URL and URI is worth noting, and although the rest of the thesis will focus primarily on the URL, the two identifiers will be discussed as distinct entities in future chapters. Equally, it should be noted that discussions involving the URL necessarily involve the URI, as a URL is a URI.

Conceptually, the URL defines a namespace that is distributed across all the servers on the web. A resource can be located if it resides at a particular location in the namespace. The URL can be extended to encode namespaces from other protocols, such as FTP (File Transfer Protocol) or NNTP (Network News Transfer Protocol), enabling resources on other information systems to easily be referenced from the web, combining their information space with that of the web's. Like HTTP, the URL treats all resources equally, and so can identify any resource regardless of its content's media-type. As such, the URL is "...the most fundamental innovation of the web, because it is the one specification that every web program, client or server, anywhere uses when any link is followed." (Berners-Lee and Fischetti, 1999, p42).

2.2.2.3 HyperText Markup Language

HTML provides a consistent markup language for the platform-neutral presentation of information. The markup language is text-based and easy to learn, two factors that have led to the web's success. HTML can be seen as the web's visual component, rendering all information resources no matter what their content. Multimedia content, such as video, and functional content, such as Java applets, can be embedded into an HTML document so that it sits next to text in the user's browser.

The original HTML specification has been improved over time, becoming richer and more functional, while a new extensible markup language, called XML (Bray et al., 2000), has been specified that is based on the principles of HTML, but whose

elements and semantics are entirely open and extensible. Indeed, the whole of the web's architecture is fully extensible: the URI can encode any namespace; the messages and semantics of HTTP can be extended; and any content in any markup language can be consistently presented. As such the web should remain the *de facto* global information system for many years to come.

2.2.3 *The Relationship Between the Web and the Internet*

In 1974, Vinton Cerf and Bob Kahn published *A Protocol for Packet Network Internetworking* (Cerf and Kahn, 1974), which specified the design of a Transmission Control Program. This was to be the forerunner of the Transmission Control Protocol (TCP), the protocol used for the transport layer of the Internet. In 1981, the TCP/IP (Transmission Control Protocol/ Internet Protocol) protocol suite was fully formalized (see RFC793 (Postel 1981a), and RFC791 (Postel 1981b)), and the term *Internet* was defined for an interconnecting network. This protocol suite was to become the *de facto* interconnecting network protocol standard, and was capable of operating above a "...wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks" (Postel, 1981a). TCP/IP is concerned primarily with internetworking heterogeneous networking systems, and does not care about the protocol layers beneath it. Users of this interconnected network can therefore communicate with other users regardless of the network technology each uses, so long as each communicating host computer can run the TCP/IP protocol stack on top of its existing network protocols.

The web uses the Internet for the transport of its resources. The physical resources of the Internet, such as its routers, provide the network infrastructure upon which the web and its services run. Beneath the Internet lie the physical network resources, such as switches, of the individual networks. Because the Internet is a data internetwork that is designed to run over any type of network, its protocols (TCP/IP) are designed simply to transport bits safely, regardless of the technology used in these physical network resources. As such, the Internet has no way of controlling the network resources over which it operates, and thus no way of enabling its services to control the network. The Internet and the physical network it runs over are seen as two entirely separate systems. Because of this, service characteristics such as Quality of Service (QoS) cannot be guaranteed, and so an Internet application that requires a certain QoS must provide its own means of compensating for bandwidth variation and latency. This does not affect most resources on the web, however, as HTTP is a simple request/response protocol riding on top of TCP/IP that enables resources to be transferred from server to client, regardless of the time it takes for the transfer to complete, or the latency inherent within the system. As such, TCP/IP provides a perfect transport for the web, as it guarantees the safe delivery of data, allowing HTTP simply to co-ordinate the downloading of resources according to their content type, without worrying about the underlying packets of data.

2.2.4 *A Philosophy of No Control*

The Internet cannot provide a guaranteed QoS because of a decision by its designers to place all the complexity of the protocols in the transport layer. In stark contrast to the telecommunications model, which seeks to control every element in a network, the Internet assumes that the network layer is inherently unmanageable, and so TCP, the Internet's transport layer protocol, is a robust, reliable protocol that works over an inherently unreliable network (Tanenbaum, 1996). As such, its design philosophy is based on reliability and openness rather than control, with its open protocols designed to work across a broad variety of networks, but at the expense of understanding or controlling those networks. Its design is characterized by open standards and open access, enabling anyone to use it or develop for it, a philosophy that ultimately led to the development of the web. Indeed, this philosophy is shared by the web, which provides a platform for service and information provision in the form of web sites. The web's standards are also entirely open, and can be extended and used by anyone without license, while information provision is unrestricted and uncontrolled. This is central to the philosophy of the web's creator, Tim Berners-Lee, who, when designing the web, felt that:

"...for an international hypertext system to be worthwhile...many people would have to post information...That meant that anyone (authorized) should be able to publish and correct information and anyone (authorized) should be able to read it. There could be no central control" (Berners-Lee and Fischetti, 1999, p41-42).

As the following sections discuss, however, this philosophy has made the web what it is today, but at the expense of providing a very shaky foundation for what the web will become tomorrow. More specifically, this philosophy has had an adverse effect on the web's ability to manage its own information, for it is difficult to manage anything without control.

2.3 *Managing the Web's Information*

All content in an information system, whether it is text, images, movies or sound, etc., encodes information in a specific media-type. In order for this information to be used effectively by the system's users, the system must provide mechanisms that structure the information and enable it to be managed. In a system the size of the web, effective management is even more crucial because of the diversity of the information, and the enormous amount of it.

There are many different components to information management, but for the purposes of the research described in this thesis, three components are key. Specifically, the research has focused on how the web:

- identifies information;
- structures information;
- retrieves information.

These three components are crucial to effective information management in a global information system such as the web, as without them, the information would be inaccessible, meaningless and irretrievable. The philosophy underlying the web's design does not provide great support for these components, but as the following sub-sections show, they do exist, either through the web's own architectural mechanisms, or through external third party services.

2.3.1 Identifying Information

The web's architecture specifies the URI as the mechanism through which information is identified. The URI is a generic syntax for naming all types of resources on the web, and so provides the web with a namespace. The vast majority of web resources, however, use the URL, which, as was discussed in section 2.2.2.2, is a human-readable subset of the URI that identifies a resource through its location. More specifically, the URL locates a resource via a combination of the Internet's Domain Name System (Mockapetris, 1987a, 1987b) and the server's own file system. In this way, the namespace of the URL comprises the namespace of the web server and the namespace of the Internet's DNS, giving the web server partial control over the location (and thus the identity) of the resource.

The namespace provided by the URL has become the *de facto* namespace of the web.

As defined in Berners-Lee et al. (1994), the syntax of the URL encodes:

- the protocol scheme to be used to download the resource (usually http://);
- the domain name of the host server (e.g. www.aserver.com);
- the file path of the resource (e.g. /library/);
- the file name of the resource (e.g. index.htm).

The Domain Name System (DNS) is used to map the server's domain name, which is encoded within the URL, onto its IP address in order that a TCP/IP connection can be made to the server. Once achieved, the resource can be found by the server through the file path and file name components of the URL. As such, it is the combination of the name of the server hosting the resource, and the file path within the server, that uniquely identifies the resource and largely defines the URL's namespace.

Another URI scheme is the Uniform Resource Name (URN), which identifies a resource independently from its location (Sollins and Masinter, 1994). Note that technically a URL is used to *locate* a resource, whereas a URN can be used to *identify* a resource (Berners-Lee et al., 1994); that is a URN persistently labels a

resource with an identifier, even when that resource ceases to exist (Berners-Lee et al., 1998). However, the concept of the URN has never been implemented, and so in practice the URL is used to both identify and locate a resource.

The URL is encoded in HTML using an *anchor* element, an example of which is:

```
<A HREF="www.microsoft.com/index.htm">Click Here</A>
```

The anchor element provides a visual representation of the hyperlink, and encodes the URL in its *HREF* attribute to both identify the linked resource, and locate it. As has been said, HTML is the web's markup language, and its anchor element can provide textual information about the hyperlink (such as 'Click Here' in the example above), or be embedded in other, richer forms of information, such as images, so that a click on the image will download the linked resource. In this way, the information presented to the user identifies the content of the resource to that user, while the URL embedded in the underlying anchor element identifies and locates the resource to the underlying system.

The hyperlink is part of the core architecture of the web. New specifications have recently been proposed to extend its functionality in the form of the XML Linking Language (DeRose et al., 2000) and XML Pointer Language (DeRose et al., 2001). These provide extensible semantics to the hyperlink, with XML Linking Language (XLink) defining a syntax for asserting and, more importantly, characterizing explicit

relationships (i.e. hyperlinks) between two or more resources (Maler and DeRose, 1998a), and XML Pointer Language (XPointer) enabling elements, character strings or other parts of an XML document to be referenced regardless of whether they carry an explicit identifying attribute (Maler and DeRose, 1998ab). Essentially these proposals allow the semantics of the relationship between two resources to be defined, and refine the granularity of the hyperlink's target to that of individual elements within a resource. However, the specifications of these proposals have yet to be completed, and they are not widely used on the web. As such, they will not form part of this discussion, and the rest of this thesis will treat the hyperlink as it is defined in Berners-Lee and Connolly (1995) as "...a relationship between two anchors".

2.3.2 Structuring Information

For an information system to be useable, its information must be structured such that it can be understood by either human or machine, depending on the nature of the system. Without structure, information becomes noise, with no form, no meaning and no use. The web primarily uses HTML to structure its information in a human-readable form. Content within a HTML document is formatted using HTML tags, which tell a user's browser where to place specific items of content, such as text, images, tables, etc., and in what style to present it. In this way, a HTML document looks attractive to the user, and can be easily read if designed well.

However, the web's architecture is less effective at structuring its information for machine use. Search engines must infer the meaning of a HTML document from its text in order to determine its relevance to a user's query, as the HTML elements are primarily concerned with the style and presentational layout of a document, rather than with the semantics of the document. In order to provide more machine-readable semantics, new markup languages based on the syntactical rules of XML, such as the Resource Description Framework (Lassila and Swick, 1998), have been designed, which encode the semantics of a document into a machine-readable description through the use of meta-data (i.e. data about data). The meta-data can be read and understood by a software agent such as a web spider (see sub-section 2.3.3), providing it with accurate information about the meaning of the web document. As such, just as HTML defines a common way of structuring information for the user, so XML and its associated markup languages define a common way for information to be structured for the machine. Note, however, that XML has been defined according to the philosophy of the web, and is an *extensible* markup language. As such, it only defines the syntax with which new markup languages must be created, and allows the structure and semantics of the language to be defined according to the needs of the user.

As well as structuring information within individual documents, an information system must also structure its information across the whole system. Collating information in this way enables the user to retrieve it easily, and to query the system in order to access information effectively, or to retrieve information from multiple

sources. However, the philosophy underlying the web's design is such that system-wide mechanisms, including those that provide information structure, cannot be part of its architecture, as they require system-wide organization and control of a kind that cannot effectively be provided in a federated system. As such, the web's architecture does not support such system-wide information structure, and so limits the effectiveness of information retrieval (see sub-section 2.3.3, below), and prevents the user from performing system-wide queries (Pitkow and Recker, 1994). Information exists within web sites as part of those sites only, and although an individual site can provide its own tools to enable its information to be queried, it cannot be queried against other information on the web. As such, information is structured on the web, but only at the level of the individual resource. Third party services may provide an alternative, permitting users to seek out information across multiple web sites, but such a service must be aware of those specific sites in the first place, either through registration or web crawling (see the following sub-section). The web, therefore, may be a vast reservoir of information, but it is disparate information that is far removed from the ordered world of the database.

2.3.3 Retrieving Information

Because the web's architecture does not provide system-wide structure to its information, it follows that information retrieval is not specified within its architecture either, as this too requires system-wide organization. The only defined mechanisms for a user to retrieve information is through navigation to appropriate web documents via hyperlinks, or using a web browser's navigational features, such

as the forward and back buttons. These mechanisms are far from effective for a system the size of the web, however, as the number of hyperlinks is vast, and the navigational features are ineffective at best. As such, it has been left to independent third party service providers, such as the search engine companies like AltaVista, Google, and Lycos, to provide effective information retrieval services, which work on top of the web's architecture.

2.3.3.1 Overview of the Search Engine

Search engines can be classified as either an index or a directory. An *index* works by storing and indexing every single word of every HTML document, and comparing them with the words in users' queries to produce relevant documents. A *directory*, in contrast, provides a hierarchical organization of documents, each classified according to a specific subject heading (Inktomi, 1996). The user can navigate through this structure using their own relevance heuristics in order to locate a document that satisfies their information need. Because of the lack of information structure within a document, however, it is difficult for a directory to correctly classify its documents according to a specific subject heading automatically, and so companies such as Yahoo pride themselves on classifying every one of their 1.8 million documents manually (Sullivan, 2000a).

To the end user, a search engine is no different from any other web site. The user navigates to the search engine, and is presented with the content of the site. The user types in a set of keywords, and the search engine presents a list of relevant web sites

that match the keywords. The user can then determine the relevance of each site to their specific information need, and navigate to that site from the search engine's web page (Figure 1).

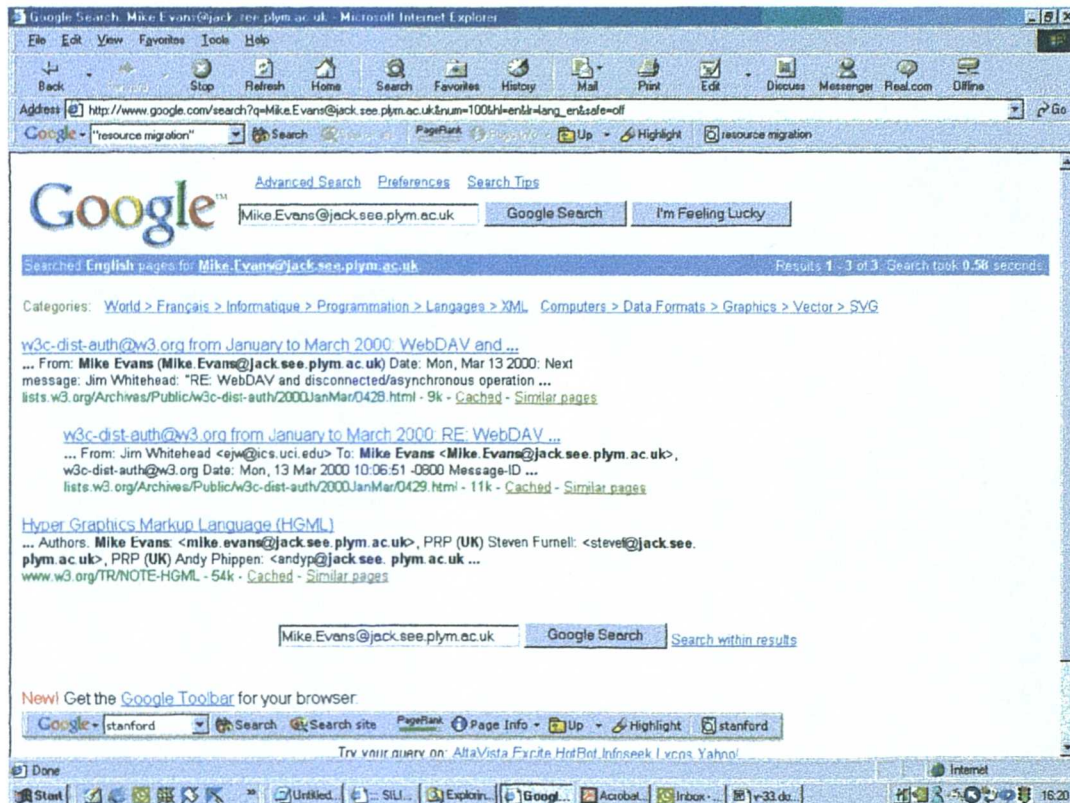


Figure 1 - Screenshot of the Google Search Engine

In order to provide a comprehensive service, a search engine must index or classify all the words in a document from as many documents as possible. The extent to which the search engine covers the web defines its size (measured in number of documents indexed), with popular search engines competing to become the biggest search engine on the web. To do this, most find their documents through the use of a software agent called a *web spider*, which automatically navigates across web

servers, indexing the information it finds in any web document it comes across. A spider performs three functions (see Pallmann, 1999, p143):

- Discovery – locate specific web sites of interest;
- Site Crawling – traverse the site to determine the resources it contains;
- Indexing – glean descriptive information from the resources within the site.

Usually, the information from the resources is added to a search engine's database, which can then be queried by a user in order to locate a specific resource that contains information that the user needs. If the web had any kind of system-wide structure to its information, a spider would not be needed. As it is, they are an essential component of current search engines, and are highly sophisticated programs, capable of searching vast numbers of web resources every day. For example, AltaVista's (www.altavista.com) spider, called 'Scooter', collects data on over 6,000,000 HTML documents a day (Schwartz, 1998), while Inktomi's (www.inktomi.com) spider indexes an impressive 10,000,000 HTML documents a day (Chakrabarti et al., 1999a). One way in which a spider discovers appropriate web sites is by extracting the URLs from hyperlinks in HTML documents, and using them to download the associated resource. If that resource is an appropriate HTML document, its words are indexed. If it contains hyperlinks, the URLs are again extracted, and the spider moves on again to the associated HTML documents. In this way, the spider is said to crawl across the web, indexing web documents as it goes.

In order to provide an accurate service, a search engine must employ a set of heuristics in order to determine the *relevance* of a document to the keywords typed in by the user. The heuristics vary in their sophistication according to the different search engines in which they are used, and are one of the main differentiating factors of the different search engines available. Google (<http://www.google.com>), for example, uses a sophisticated heuristic called *PageRank* (Brin and Page, 1998), which ranks a HTML document's authority based on the number of hyperlinks that point to it, or the number of documents that point to it which themselves have a high PageRank value. Google then determines the relevant HTML documents that match the user's query according to traditional relevance heuristics (e.g. keyword matching), and uses the PageRank value to rank those that are deemed relevant according to their computed authority. However, if the web provided more structure to its information, a search engine would not need such complex relevance heuristics to infer the meaning of the document from its text.

2.4 *The Information Management Dichotomy*

The philosophy behind the design of the web has made it very difficult for its information to be managed. The web has no central authority controlling who can provide information or a service. There is no concept of service modelling on the web, as there is in a telecommunications network (DOLMEN, 1996a), and no uniform way of providing a service. Effectively, anybody can publish anything, and so all of the web's content has been provided by independent third parties who require no permission to provide their information, and are subject to no regulatory

controls. In this way, the only control over information and service provision on the web comes from market forces. The search engine companies illustrate this, with user demand highlighting the need for an information retrieval service being quickly met by competing companies rapidly developing and deploying competing search engines. However, even these forces only control information that is *expensive* to provide. Information that is inexpensive, such as family photos intended purely for a small circle of people, is not subject to these forces, and with HTML being so easy to use, and web servers providing their information hosting services for free, there is effectively no control over information provision. As such, the lack of information management in the web's architecture leaves it exposed to an uncontrollable explosion of information of questionable value.

However, the issue of effective information management is complicated by the effect it can have on a system's user population. A global information system needs to ensure that enough users use it and publish information on it in order for it to be viable, but the specific type of information management policies employed by the system can have a dramatic effect on the system's adoption. Too much management can provide the operator of the system with too much control, and will tend to discourage the system's adoption. Conversely, too little will render the system useless.

2.4.1 Gatekeeping

The reason that managing information can provide the manager with too much control is that it turns the manager into a censor. A censor is defined as:

“an official authorized to examine printed matter, films, news, etc., before public release, and to suppress any parts on the grounds of obscenity, a threat to security, etc.” (COD, 1990).

Rephrased, a censor is an agent with enough control to suppress information according to his or her own individual criteria. Thus, whoever operates a global information system has the power to suppress whatever information they like according to whatever criteria they choose, once they have the appropriate information management structures in place. It is this uncomfortable fact about information management that can prevent users from adopting a system in large numbers.

Such censorship has been termed *gatekeeping* (Levinson, 1997), and has been in existence since information itself. This is self-evident, however, as to give information any value, it must be selected according to its quality, and somebody has to decide which information is of higher quality than other information. That person is the gatekeeper. In this way, the gatekeeper manages information, and makes the information selected more effective in conveying the intended meaning. Traditionally, the content provided by all media has been controlled by a few individuals (usually ‘media-barons’ such as Rupert Murdoch or Robert Maxwell)

who are in a position to control the media that they own. These individuals can influence the editing of their newspapers, magazines, books, etc., and so act as gatekeepers, effectively selecting the information presented to the reader according to their own biases, politics, or opinions (Levinson, 1997). The information is well managed and informative, but the reader can only choose to read whatever information the gatekeeper selects for them. As such, a balance must be struck: too much management leads to propaganda, an unwanted state of affairs for any information system; too little, however, and no effective meaning can be conveyed.

From the discussion in section 2.3, the philosophy of the web clearly leans towards the open, almost anarchic style of management. However, the web is not the only information system, and others, such as the service architecture defined by the telecommunications companies (DOLMEN, 1996a), or the online information providers that predate the web, favour the more dictatorial, gatekeeping approach. There is no censor on the web, as no one controls it. In contrast, the models of the service architecture and the online information providers give the network operators so much control over the information on their systems that they do become censors. Each approach has its strengths and weaknesses, but as the following sub-section demonstrates, the anarchic competitor will always beat the managed architecture.

2.4.2 Case Study: Proprietary Online Information Providers

Before the web enjoyed its present popularity, there existed a set of proprietary online information service providers that included CompuServe and America OnLine

(AOL). These online service providers provided an information browsing service similar to the web, but they adopted a different model of information management, preferring central control governing service access and provision, instead of the open, federated model of the web. As such, in order to use one of these online services, a user was required to use the service provider's own network, complete with its own network infrastructure, and to browse for information using a special browser that would only work with this network. Information and communication services were thus bundled together (Gillet and Kapor, 1997), giving an online service provider (OSP) complete control over who had access to their system and who provided information. As has been said, the OSPs predate the web, but their existence also coincides with the birth of the web. As such, these service providers provide a real life case study of the impact the open model of the web has on a system based on the gatekeeping approach.

Before the web, the OSPs could generate high revenues with a high profit margin, as they could control both access to their information, and the provision of the information. Each service provider offered a closed system with its own network, and limited or no interoperability between systems, effectively locking the user in to their system. However, the web is a fundamentally different competitor, not least because it is not owned or controlled by any one person. In particular, the web has a fundamentally different business model: it is free, offers no barriers to entry, no control over the system, and separates information providers from connectivity providers. Thus, the web provides three distinct advantages over the OSPs:

1. *Choice* - To use an online service, you must use the online service provider's software. With the web, however, the user can choose both the software vendor and the type of application they wish to use, and can choose to adopt new services, such as Internet telephony, only if they want to, and without being restricted to a particular vendor or a particular service provider (Gillet and Kapor, 1997).

2. *Freedom* – With no barriers to entry, anyone can provide any information on the web without fear of censorship, and with little cost. Over time, the web will therefore provide more information, as it can provide information that is too expensive to be deployed on an online service, or that the online service provider refuses to publish (i.e. censors). Thus, over time, the web will attract more users, as it will have more information, which, in turn, will attract more information providers, setting in motion a virtuous circle of information provision and access.

3. *Competition* – Anyone can innovate on the web, and anyone can provide information. This leads to ferocious price and information competition amongst a diverse array of providers, improving the diversity of the information to a degree not shared by users of the online services (Gillet and Kapor, 1997).

Ultimately, the web shifts control over information provision from the OSP (or, in the case of the service architecture, the telco) to the information owner, putting severe pressure on the revenues that the OSPs receive from the information owner (Gillet and Kapor, 1997). Since the advent of the web, there has been a huge shake-up in the online services market, with AOL taking over CompuServe, and ultimately finding that the only way it can compete with the web is to become a gateway to it. Users of online services do not want to be precluded from the web, and users who use the web largely have no need for a proprietary online service. As such, the remaining OSPs have been forced to reduce their prices and effectively become *Internet Service Providers* who also offer their own private content, while their information browsing service simply becomes another (albeit large) web site that requires a subscription payment for access. Essentially, the OSPs, although large companies in their own right, have become niche players in the information browsing market, with the web defining the boundaries of and the platform for that market.

To a large degree, this process was inevitable. Metcalfe's Law states that "the value of a network grows as the square of its number of users" (Metcalfe, 1996), and a network such as the web, which neither censors its information nor discriminates against its users, will always have more users than a network that does. From the perspective of a network, then, censorship is the wilful restriction of the network's user base, and so from Metcalfe's Law, it can be seen as a restraint on the network's value. As such, a network employing the gatekeeper approach to information

management will always be less valuable than a network that is open and censor-free.

However, the gatekeeper approach is not only commercially impractical when competing with the web, it is also socially undesirable, particularly in a *global* information system. Information deemed unpublishable in one country is perfectly acceptable in another. Further, letting the user decide on the service they wish to receive empowers them, providing them with a new tool for self-expression. Effectively, it turns the web into a common platform for the exchange of ideas, which, if adopted by enough users, can have a profound impact on society, if only in speeding up the dissemination of ideas. The future of user information systems, therefore, lies with the web and its open model of information management.

2.4.3 Internet Anarchy

Using the case study of the online service providers, the argument has been made that a system with rigid information management structures in place, and which provides control over information access and provision, will not be able to compete with the web. The web is a decentralized, federated system that gives no one the power to control information, and it is this fact alone that guarantees its success, and ensures it will always beat a gatekeeper network. However, it is also this fact that can lead to the web disintegrating into a sea of noise. Specifically, the lack of central control has led to the development of an ad hoc architecture, whose central features such as its navigation mechanisms have been developed in response to user need,

rather than through a well-designed plan. That this architecture is able to support over 300 million users is testimony to the power of such a flexible approach, but it has also led to “Internet anarchy” (Raatikainen et al., 1998), with high quality information becoming drowned out by low quality information, and flaws are beginning to appear in the web’s architecture that could potentially prove fatal.

When Metcalfe first determined the value of a network, which Gilder later described as a law (see Gilder, 1993), there was perceived to be no upper bound to a network’s value, but there was also no information network the size of the web. As such, there is tentative evidence (although no empirical data) that there may be an upper bound to Metcalfe’s law, and that beyond this limit, adding users *diminishes* the network’s value (Windrum, 1999). The hypothesis suggests that users add information and thus value to a network, but that information that is not relevant to a user should be classed instead as *noise*. As such, the more users that use the network, the more information that is published, but the less likely it becomes that any individual item of information is relevant to a particular user. Beyond the threshold, the information content of the network actually *decreases* with each additional item of information.

Other information systems that are based on the gatekeeper approach use the web’s architecture as a model of what can go wrong with a global information system, despite its obvious success. For example, the telecommunications companies have envisaged themselves as providing the *de facto* global information system for many years and their research efforts have culminated in the design of the service

architecture. One such service architecture, OSAM (Open Service Architecture for Mobiles (DOLMEN, 1996a)), which was developed as part of the European DOLMEN project (Service Machine Development for an Open Long-Term Mobile and Fixed Network Environment (DOLMEN, 1995)), and funded by the European Union's ACTS Programme (Advanced Communications Technologies and Services, 1994 – 1998), used the web as a benchmark global information system, analysed the flaws in its architecture, and then sought to circumvent them in its own architecture through the use of rigid information management mechanisms that actively enforced gatekeeping (see Raatikainen et al. (1998)). As such, the web was regarded as no more than a good effort, whose limitations were gradually undermining its effectiveness.

However, service architectures are still in the research phase, with no commercially deployed network based on the model in existence, and no matter how effective such a network is at managing its information, it will never reach the number of users necessary for it to compete with that of the web. Equally, the web has a flawed architecture (Raatikainen et al., 1998), and one that will cause more problems as its size continues to grow. As such, a dichotomy now exists at the heart of the development of a global information system the size of the web: specifically, gatekeeping prevents the system from maturing, especially when a non-gatekeeping system is a competitor; whereas non-gatekeeping makes a maturing system degrade into noise. Ultimately, the only solution to noise in a large system prevents it from becoming large enough to suffer this problem in the first place. Unless this

dichotomy is resolved, the only global information system that can exist is one that is doomed to degrade into noise under the weight of its own information.

2.5 *Summary*

This chapter has introduced the web and the protocols that form the basis of its architecture. The web has been shown to be a fundamentally different system from traditional information systems, with a very open architecture, and a flexible philosophy that allows anyone to access anything, publish any kind of information, and develop any kind of service.

The chapter has also introduced the topic of information management, describing how it is implemented on the web, and in alternative systems that adopt a more controlling approach. The chapter has revealed the information management dichotomy, which conspires against a global information system regardless of the approach to information management that is adopted, and acts to limit the effectiveness of any system unless it can be resolved. As such, the dichotomy lies at the heart of the research conducted as part of this research programme, which has attempted to resolve it for the benefit of the web through the development of a novel model for managing information flow. The model will resolve the dichotomy by fixing some of the web's flaws in such a way that its open philosophy is not compromised.

Part of the work described in this thesis has been funded by research for the DOLMEN project, which has provided rich insights into the flaws of an information system, and the different approaches to information management that can be used. The following chapter will therefore use research from the DOLMEN project, in which the author was a participant, and also from other studies that have been conducted, to provide a comprehensive analysis of the web's flaws in order to identify their exact scope and nature.

3. Flaws in the Web's Architecture

Weak information management has enabled the web to grow, but at the expense of its overall effectiveness as an information system. As the web has matured, certain flaws in its architecture have been revealed, which are now beginning to pose a threat to the coherence of the system. This chapter provides a comprehensive analysis of these flaws in order to determine the extent of the web's problems, and describes how the different solutions that have been proposed during the web's lifetime have all failed.

3.1 Introduction

The web is not a perfect system, and many studies have been conducted that examine its faults. For example, Cockburn and Greenberg (1999) have examined the limitations of the navigational tools provided by browsers, particularly the Back button; Park et al. (1997) have shown how the network traffic produced by the web has a serious adverse effect on network performance; many studies, such as Chakrabarti et al. (1999a), Lagoze (1997), and Lawrence and Giles (1999) report on the inefficiencies of search engines, particularly the problems they face in having to index over one billion web resources; and an equally large number of studies, such as Notess (2000a), and Koehler (1999), have reported on the problem of broken hyperlinks on the web.

A comprehensive analysis was provided by the DOLMEN project, which analysed the flaws in the web's architecture in great detail in order that it could provide a more architecturally sound model of information management for its service architecture. The DOLMEN project identified many flaws, most of which derive from early engineering decisions that did not anticipate the scale of user adoption of the web (Raatikainen et al., 1998). A detailed discussion of these flaws can be found in Raatikainen et al. (1998) and DOLMEN (1996b), but an overview is presented here:

- *Poor Information Retrieval*

Information on the web is unstructured and is not categorized, and although search engines provide an information retrieval service, they are isolated from one another, and there is no standard way to find their servers and access them (Raatikainen et al., 1998).

- *Poor Navigation*

A hyperlink points to any type of resource, which allows the web to host information of all media types. However, there is no way for a user to discern the type of resource that a hyperlink may point to. Thus, the user cannot tell whether a hyperlink points to a product, an individual, or information, and so must manually click each hyperlink in order to determine its associated resource's type. This inefficient process is compounded by the fact that many hyperlinks are old and out of date, or simply point to resources that no longer exist (Raatikainen et al., 1998), a process known as *link rot*.

- *Poor Information Structure*

The web provides no standard way to organize and maintain its information, with each web site using its own approach to data management (Raatikainen et al., 1998). Although XML provides a standard for data formatting and interchange, there is no way to cross-reference multiple web sites, and so query multiple web servers.

Some of the flaws identified by DOLMEN, and the problems analysed by other studies such as those discussed above, can be dismissed as inconveniences - either features that would be nice but are not essential, or problems that will be solved over time as technology inevitably improves. As such, no architectural improvement is required to address them, as third party services or general improvements in the web's infrastructure will resolve them anyway. However, there are three specific flaws that have beset the web since its origin, and although they have been treated as irritants up until now, they have the potential to seriously cripple the web and render it completely ineffective. Specifically, these flaws are:

- *Link rot* – the web has no way to update hyperlinks that point to missing resources, causing the hyperlinks to 'rot' over time;
- *Shrinking namespace* – the names used to identify resources are running out, leaving only meaningless, unmemorable names with which to identify resources;
- *Increasing noise* – with no informational structure, and poor navigational tools, it is becoming increasingly difficult for users and search engines to distinguish high quality relevant information from the surrounding noise.

These three flaws pose the greatest threat to the web's continuing success, and must be solved soon if the web is to continue its exponential growth. However, as was shown

in section 2.4, conventional solutions rely on system-wide information management that effectively censors the web. As such, the three flaws represent the concrete realization of the information management dichotomy: the web's information must be managed effectively in order to solve the three flaws, but it cannot be managed too effectively or it will alienate its users by giving too much control to a central organization. This is the key problem that the research described in this thesis attempts to solve: to provide a means of solving these three flaws in the web's architecture in a way that is compatible with its open philosophy.

The following sections provide a detailed examination of the root cause of these flaws, revealing why they are so dangerous to the web, and discussing some of the solutions that have been proposed, but which have not worked.

3.2 *Link Rot*

Link rot is the process by which the resources that are referenced by hyperlinks in another resource are deleted over time, leaving the hyperlinks pointing to nothing. From the perspective of the resource that contains the hyperlinks, they can be seen to effectively rot over time, as the referenced resources are gradually deleted. It is characterized on the web by the HTTP *Error 404* (Fielding et al., 1999), which is returned to the user in place of the expected resource. Its cause is the result of the web not having *referential integrity*; that is, the integrity of its references (the hyperlinks) is not guaranteed, allowing the references to break whenever the resource changes location or is deleted. A system has referential integrity if a resource is guaranteed to

exist for as long as outstanding references to the resource exist. However, the web cannot guarantee this, as the hyperlinks are entirely independent from the resources that they reference. This leaves the resource owners entirely unaware of the existence of the hyperlinks, making it impossible to determine how many references to a resource exist (Ingham et al., 1995).

3.2.1 The Cause of Link Rot

The root cause of link rot on the web is the misuse of the URL in identifying a resource. It is a common misconception that a URL identifies a resource, when in reality it defines a location; the IETF has defined the URN for identifying resources. As such, a URN persistently identifies a resource throughout its lifetime, whereas the URL identifies the place where the resource may reside (Sollins and Masinter, 1994). If the resource changes location, it necessarily adopts a new URL that identifies that location, but its URN remains unchanged. As such, link rot would be solved if hyperlinks used the URN rather than the URL, because of the URN's persistence and location independence. At the time of writing, however, the concept of the URN is still experimental, and there is no working mechanism for resolving the location of a resource from its URN. A URN may therefore identify a resource, but there is no way it can be used to determine the resource's location. This leaves the URL as the only way of identifying a resource on the web, even if it is through its location rather than its name. As such, using the existing web architecture, the simplest approach to solving link rot is to constrain a resource such that it only occupies a single location throughout its lifetime.

As the following list shows, however, there may be many reasons why a resource has to move:

- *Change of server* – a resource might migrate if an overloaded web server is replaced by a more powerful machine, requiring the resource to migrate onto the new server (Ingham, 1996). Equally, the resource owner could decide to use a different server hosting company, which uses a different domain name and thus affects all the resources' URLs (this is particularly valid for home users using free web hosting companies such as Freeserve, where the name of the host must be part of the URL of the resource).
- *Resource is archived* - and the owner has a specific, lower powered machine for archived resources (Berners-Lee, 1998).
- *Bankruptcy* – the resource owner goes out of business, and the web server hosting the resources is sold, forcing the resource owner to migrate the resources.
- *Resource reaches expiry date set by server* – the resource may be deemed too old to be worth storing on a web server, which could insist that the resource owner move the resource before it is deleted.

- *Poor internal information management processes* – some resources in a web site may be confidential or out of date, but due to the size of the site and the difficulty tracking the attributes of each resource, the owners perceive it as safer to limit the accessibility of the site to internal users only, and so move it to a more secure server (Berners-Lee, 1998).

However, even if a resource was to remain on the same server, its URL could still change. For example, this could occur due to:

- *Change of resource ownership* – a competitor buys a web site, and does not wish to use the old competitor's name in any URL. Thus, the existing resource's URL must change, even if the web server hosting it does not (Berners-Lee, 1998).
- *Change of company name* – a company re-brands itself, and not wishing to use its original name, changes all of its URLs to reflect the new name.
- *Any identifier based on the DNS will always be unstable* – because of the hierarchical nature of the DNS namespace, identifiers based on it, such as the URL, tend to reflect administrative hierarchies (e.g. www.university.ac.uk/faculty/school/course/year/semester/module/week1.htm) However, such hierarchies change frequently compared to the lifetime of the resource (Moore, 1996).

Identifying a resource through its location almost guarantees a loss of referential integrity, but it is a function of both the identifier and the name service used, rather than the actual information system. Architectures that employ strong information management techniques, such as the DOLMEN project's OSAM, do not suffer from link rot as they employ name services that can guarantee referential integrity if it is required. OSAM, for example, defined its own naming service, which ensured that CORBA objects (OMG, 1995) containing the functionality of the system could still be located after a mobile terminal that hosted them had moved across GSM base station domains, and thus had its IP address changed (DOLMEN, 1997). In contrast, the web is left with the problems of having no mechanism for referential integrity; resources that are identified through their location; a naming scheme that cannot be used to locate a resource; and resources that will always move location. Under these circumstances, link rot will remain a very real problem, which, as the following subsection shows, may become terminal.

3.2.2 The Damaging Effects of Link Rot

The effects of link rot on a system can be compared to that of light from a star reaching a person on Earth. The light from a star may have travelled so long to reach Earth that the star from which it originated may have died millions of years ago. As such, the stars in the night sky are a physical representation of the universe as it was, not how it is. In the same way, link rot leaves hypertext documents, and particularly search engines with the enormous number of hyperlinks that they reference, in a

similar state: a search engine can be seen as a representation of the web as it was, not how it is, as link rot will have caused some of the links to point to resources that are no longer there any more.

Link rot at its worst will render the web useless, as no hyperlink will point to an associated web resource. In this worst-case scenario, the whole hypertextual structure of the web breaks down, which is why link rot is so dangerous. Moving away from this extreme, link rot still threatens the web over time, as if it is allowed to proceed unchecked, there will eventually be more dead hyperlinks than live ones. However, link rot of any degree makes the web sub-optimal and reduces the user experience. Although no empirical data exists that determines the level of link rot required before a user gives up using the web, studies by GVVU in the USA (1997-8) have found it to be the user's second biggest irritant (in 1997, 49.90% of users cited link rot as one of the worst features of the web (GVVU, 1997); this rose to 57.7% in the following year (GVVU, 1998)). The following lists some of the problems that link rot in even its mildest form causes:

- Reputation of the resource provider is tarnished (Ingham et al., 1995).
- Direct loss of revenue for both the migrated resource owner, whose resource can no longer be located, and the hyperlink owner, whose site's reputation is tarnished (Harris, 2000).

- Brand damage, if the sites affected by link rot are e-commerce businesses (Harris, 2000).
- Loss of productivity, particularly for large web sites with thousands of hyperlinks to maintain (Harris, 2000).
- Unreliable referencing for scholarly citation (Kahle, 1996).
- Lost digital history, as a deleted document is gone forever (Kahle, 1996)
- Compromises the services provided by librarians, as it imposes a huge burden on catalogue maintenance (Shafer et al., 1996).

Clearly, link rot poses at best an irritant to the web user and resource owner, and at worst a distinct threat to the future of the web. To determine how great a threat, the following sections present a comprehensive literature search on existing studies that have analysed the problem, and discuss an experiment that was performed as part of this research in order to provide new empirical data.

3.2.3 Measuring Link Rot in the web

To measure the level of link rot, it is necessary to provide accurate statistics for the number of hyperlinks on a web page that are broken (link rot incidence); the number

of web pages that contain broken hyperlinks (link rot prevalence); and the average life span of a web resource.

3.2.3.1 Link Rot Incidence

Few studies have attempted to determine link rot incidence. Sullivan (2000b) selected 200 web pages at random from the AltaVista search engine, and determined the number of broken links in each, reporting that the percentage of broken links in each document averages 5.7% (Sullivan, 2000b). Notess (2000b) has studied the incidence of link rot in the major search engines, sampling the first 100 links returned for each of three separate searches in the major search engines. He reports that AltaVista is currently the most affected by link rot, with some 13.7% of links broken, with Excite (www.excite.com) and Northern Light (www.northernlight.com) recording 8.7% and 5.7% respectively. Finally, Lawrence and Giles (1999) took various measurements of the major search engines, and found that link rot in 11 of the major search engines varied from 2.2% for HotBot (www.hotbot.com) to 14% for Lycos, with an average of 5.3%, which would seem to confirm Sullivan's results.

3.2.3.2 Link Rot Prevalence

Even fewer studies have attempted to determine link rot prevalence. From the same study described above, Sullivan (2000b) has reported that link rot affects 28.5% of all web resources, but other studies do not report link rot prevalence.

3.2.3.3 *The Life Span of a Web Resource*

The life span of a web resource (that is, the length of time in which it is accessible on the web in any one location) has been measured in various studies, with Kahle (1997) reporting 44 days, and Gwertzman and Seltzer (1996, as cited in Pitkow (1998)) estimating 50 days (confirmed through comparison with other studies in Pitkow (1998)). Other results come from Koehler (1999), whose year-long study monitored 343 URLs, selected at random from the WebCrawler (www.webcrawler.com) search engine's index, and found that 25.3% of the resources had died, giving the half-life for a web *site* as 2.9 years. These results were confirmed by Lawrence et al. (2001), whose analysis of hyperlinks that were provided as references in scientific papers in the ResearchIndex database (www.citeseer.com) showed that 23% of the hyperlinks no longer worked after one year.

Alarming, however, academics and non-academics alike use Kahle's figure of 44 days as if it is a matter of fact (see for example CyberMetrics (2000), Ashman and Davis (1998), Harris (2000), Pearson (2000), and McNamara (2000)), when in reality, it does not measure the life span of a web resource at all. Presumably, the fact that Kahle's article appeared in *Scientific American* (see Kahle (1997)) has given the article a certain gravitas, such that it is seen as a more authoritative study than those whose results are published in other journals, and is therefore more widely cited. Further research reveals, however, that the figure of 44 days does *not* represent the life span of a web resource.

Discussing his proposal for an archive of the Internet (since developed as the Alexa archive (www.alexa.com)), Kahle (1997) states that "...estimates put the average lifetime for a URL at 44 days." However, he does not explain the method that was used in reaching this figure, and nor does he quote its source. Personal communication with Kahle (Kahle, 1999) as part of this research has revealed that he simply used the figure from work done on the Harvest project by Peter Danzig and his colleagues (Chankhunthod et al., 1995), without quoting its source in the Scientific American article. More revealing is the fact that another of Kahle's web sites, www.archive.org, contains a first draft copy of Kahle (1997), in which it is claimed that "...the average lifetime of a document is 75 days and then it is gone" (Kahle, 1996). Upon reading (Chankubnthod et al., 1995), it is clear where the two figures come from: 44 days is the figure quoted by Danzig as the mean lifetime of all web resources, whereas 75 days is the mean lifetime of HTML documents. However, Danzig's work was involved in developing a web-wide cache, and the figures of 44 and 75 days represent the period that a web resource remains *unmodified*, not the period in which a web resource remains *accessible*. Danzig was attempting to determine whether or not there was an average period of time in which web resources remain unchanged such that he could set a default Time To Live value for his caches, and so prevent the caches from going stale. He was not trying to determine the life span of a web resource at all. Kahle himself accepts that the figure is now "no longer valid" (Kahle, 1999), claiming that an internal study at Alexa shows that 6% of HTML documents *change* in 3 months and adding the caveat that "this does not mean that they did not disappear" (Kahle, 1999). However, this has not altered the perception in

the web community that 44 days is the true value for a web resource's life span, with companies such as LinkGuard actively promoting this figure (see Harris (2000)), as their entire business model rests on its customers believing its validity.

This leaves the web in a vulnerable position, as the true level of link rot in the system, and therefore of the reduced value of the web, is entirely unknown. Although other studies such as Gwertzman and Seltzer (1996) provide a different source of empirical data, they are now four years out of date, and overshadowed by the authority of the Scientific American article.

3.2.3.4 An Attempted Experiment to Determine the True Level of Link Rot in the Web

In order to provide a more accurate measurement of link rot, an experiment was designed and conducted as part of this research, that was designed to track web resources over time to see how long it took before they moved location, thus breaking any hyperlinks pointing to them. In addition, the experiment also attempted to determine the length of time before the content of a web resource changed, to see if Danzig's original figures are still valid.

The experiment involved the collection of a large sample of random links, with each link being tested periodically, and the date and time recorded if and when a link failed (i.e. the resource pointed to by the link could no longer be found). To ensure the randomness of the links, the experiment required the compilation of a database of web servers from a large list of Internet servers chosen at random. Once the list had been

compiled, the intention was to let a web crawler search through the various HTML documents on the web servers, and choose for itself a set of links at random. This would ensure no bias had crept into the link selection process. Unfortunately, however, the experiment never reached this stage, as several unanticipated side effects caused two unintentional security incidents, which forced the experiment to end prematurely. The side effects were a direct result of the experiment testing for the existence of web servers through randomly pinging IP addresses, and falling foul of the configuration settings of two remote firewalls. This led the University's Computing Service being understandably concerned that further problems could arise if it was to continue, and so the mutual decision was taken to discontinue this element of the study, despite over 700,000 IP addresses being collected. A full discussion of the experiment and the problems incurred can be found in Evans and Furnell (2000).

3.2.3.5 Determining Link Rot from the Literature

The failure of the above experiment has forced the figure for the length of time before a link can be expected to rot to be determined by the literature, which at the time of the experiment was not that accurate. The figure of 44 days used by Kahle has been shown to be the lifetime of the *content* of a resource in a cache, rather than the resource itself; that is, the length of time before the content within the resource is modified. As such, this figure must be compared with Brewington and Cybenko's (2000) figure of 117 days for *content lifetime*, rather than with any figure for *resource lifetime*, with Brewington and Cybenko's figure being the more accurate as their experiments were far more comprehensive and recent. Alternatively, the figure of 50

days derived by Gwertzmann and Seltzer's (1996, as cited in Pitkow (1998)) work has been independently verified, and is reported in Pitkow (1998), itself a widely cited paper. However, this figure is now out of date, and contradicts Brewington and Cybenko's (2000) findings, implying that the content of a resource lives for over twice as long as the resource itself. As such, the thesis will use the figures determined independently by Koehler (1999) and Lawrence et al. (2001) as the measure of a resource's lifetime, as both studies are comprehensive and recent, they agree with one another, and they are consistent with the findings from Brewington and Cybenko (2000). As such, the thesis will assume that the half-life of a web site is 2.9 years, and approximately 25% of hyperlinks will break after a year.

3.2.4 Existing Solutions to Link Rot

Link rot is the direct result of a resource either migrating from one location to another, or being deleted in a system without referential integrity. As such, in order to provide referential integrity, a system must provide some form of *resource migration mechanism*, which ensures the integrity of hyperlinks whenever a resource migrates or is deleted. This sub-section analyses the issues surrounding resource migration on the web, and examines why mechanisms designed for the web have not been widely accepted.

3.2.4.1 Resource Migration Mechanisms

Referential integrity is an important concept in distributed systems, and many types of resource migration mechanisms have been designed to support it. Most of the mechanisms operate by introducing a level of *indirection* into the system, which

usually involves mapping the name of a resource onto its location, or an existing location onto a new one. They can be classified according to five distinct approaches, each of which tends to place the indirection at a different point in the system. However, as the following list shows, each approach has its own fundamental weakness.

- *The Chain Approach*

A *forward reference* that points to the new location is left behind on the machine that the resource has migrated from. The indirection in the system takes place on the servers hosting the resources. Although arguably optimal in terms of network traffic overhead (Ingham et al., 1996), this approach can lead to forward references outnumbering resources. Various shortcut operations can limit the length of the chain of forward references, but this approach is still inherently brittle, as locating your resource is dependent upon the state of someone else's server. Also, a resource can only migrate onto a server that supports this approach.

- *The Callback Approach*

A database of all the links on the web is maintained, either centrally, or distributed across the system. Each time a resource migrates, the database is updated and calls back all documents that contain a link to the resource, enabling each document to update its links. This approach does not introduce any indirection into the system, as it attempts to fix broken links *in situ* rather than redirect any attempt to locate a resource. However, it does have problems scaling, particularly to a system the size

of the web. Specifically, the database must store all updates to servers that are down, and this would eventually overwhelm the system (Briscoe, 1997). This approach also requires the documents to be intelligent enough to remove links identified as broken, and so is not backwards compatible with the web's existing architecture.

- *The Search Approach*

Each server is aware of the identifiers of the resources that it hosts. Whenever a resource needs to be located, each server in the system is queried using the identifier of the required resource. A network-wide search must be performed, with a flooding algorithm used to guarantee that all servers are searched in the quest for the resource. Again, there is no indirection introduced, but flooding algorithms do not scale well, and so although reliable, such an approach would produce too much network traffic overhead for use on the web, and is the least optimal of all the approaches (Ingham et al., 1996).

- *The Name Server Approach*

A name server is used that maps a resource-identifier onto the resource's location. The name server supplies the location of the resource when given its persistent name. Clearly, the name server itself introduces indirection into the system, almost by its definition. The problem with this approach is that the web already has a name service in the form of the DNS, and integrating another name service into the web's architecture would require upgrading all the browsers on the web.

However, because the DNS operates at the level of the host (i.e. the server), it cannot be adapted to operate at the level of the resource (Daniel and Mealling 1997). In particular, the DNS is a read only database that cannot be updated over the network (Albitz and Liu, 1997), and although there is an experimental dynamic update specification (see Vixie et al., 1997), it is insecure, and can only be used to update the location of servers, not resources.

- *The Lecturing the User Approach*

Not a technical approach, more a philosophical one. Berners-Lee and others have argued that a URL need not break if considered thought is given to its design (Berners-Lee, 1998). However, despite the numerous technical arguments against this viewpoint, it is people who create URLs and people who are notoriously bad at consistent regular maintenance. Ultimately, as broken links on the W3C web site itself testify (e.g. the link <http://discuss.w3.org/mhonarc/w3c-tech/threads.html> on the document located at <http://www.w3.org/MobileCode/Workshop9507/> is broken), lecturing the user will be ineffective at best.

Table 1 provides a comprehensive, though by no means exhaustive, list of some of the different resource migration mechanisms that have been developed for the web, and illustrates the differences between the different approaches (the 'Lecturing the User' approach is not included in the table).

Approach	Name	Description	Advantages	Disadvantages
Name Server	Uniform Resource Names (URNs)	<p>The 'official' IETF solution. A URN is a "...persistent, location-independent identifier for Internet resources...[that] overcomes most of the problems with URLs" (Daniel and Mealling, 1997, p2). A URN persists for many years, with new name servers (termed <i>Resolvers</i>) designed to resolve URNs onto URLs. A Resolver Discovery Service (RDS) (Sollins, 1998) is used to locate the appropriate Resolver, with ideas for implementing the RDS ranging from expanding the functionality of the DNS (Daniel and Mealling, 1997); using proxy servers (Iannella et al., 1996); using LDAP servers (Cuenca et al., 1999); or using a reverse proxy server (Shafer et al., 1996). Referential integrity is ensured, as the URN remains persistent while the URL of the resource changes, whenever a resource migrates.</p>	<ul style="list-style-type: none"> • Official IETF solution, which embeds location-independent naming into the web's architecture. • Scalable solution • Guarantees referential integrity • Also uses the Uniform Resource Characteristic (URC) identifier, thus enabling a resource to be located through its characteristics as well as through its name and its location. 	<p>To support URNs, users must upgrade browsers and servers, the URLs of over one billion web resources will need to be converted (Moore, 1996), and no consensus has been reached on exactly how the resolution system should work. Daniel and Mealling (1997) proposed using the DNS to locate the required resolver, but this was discounted as the DNS's importance in basic network routing prohibits its use in experimental work (Daniel and Mealling, 1997). In addition, a URN is a machine-readable identifier, and requires mapping onto an as-yet undefined human-readable identifier for human use (Sollins, 1998). As such, the URN remains a theoretical solution.</p>

Table 1 - Analysis of Existing Migration Mechanisms (continued on following pages)

Approach	Name	Description	Advantages	Disadvantages
Name Server (cont.)	Persistent URLs (PURLs)	<p>A PURL (Shafer et al., 1996) is a resource identifier that uses the syntax of the URL, but alters its semantics. Rather than pointing to the location of a web resource, it points to the location of a persistent name server, which resolves the supplied (P)URL onto an actual URL.</p> <p>As such, the PURL is a URN that follows the syntactic scheme of the URL. In this way, the PURL remains persistent while the URL changes, but the web's infrastructure remains unchanged, as PURLs can be resolved in the same way as URLs.</p>	<ul style="list-style-type: none"> • Very easy and cheap to implement. • Does not require the client or the server to upgrade their software. • Provides an interim solution until a Resolver Discovery Service is developed. • Scalable solution. 	<p>In practice, the system is simply a dressed up version of a reverse proxy server. The 'persistent name server' is a reverse proxy that manages all requests to a set of registered web servers and redirects the request using the HTTP redirect mechanism to the appropriate web server.</p> <p>PURLs have not been widely adopted, however, because the PURL used by the user must contain the name of the name server (e.g. purl.oclc.org) rather than the name of the resource owner (e.g. www.amazon.com).</p>

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach	Name	Description	Advantages	Disadvantages
Name Server (cont.)	The Handle System	<p>The Handle system is a persistent global name service that provides a system for managing the identity of a resource that is both persistent and location-independent (Sun and Lannom, 2000). A Handle is the system's identifier, which identifies the naming authority and the local name of the resource as defined by that naming authority. The Handle system can be used to map a Handle onto a URL, a URN, or even a CORBA Object Request Broker or an email address.</p> <p>The Handle system defines a Global Handle Registry to locate local Handle services that can resolve a Handle to the appropriate location of the resource. Proxy servers and browser extensions can also be used to locate the appropriate Handle service.</p> <p>The Handle persists over time and space, but it is the responsibility of the naming authority to update a resource's location.</p>	<ul style="list-style-type: none"> • Very flexible system that can identify any type of resource, including non-web resources not usually associated with URLs. • Tight integration into the web's architecture through conformance with IETF specifications for URIs. • Can also work outside of the constraints of the URI scheme, and adopt any naming authority's identification scheme (Sun and Lannom, 2000) • A single Handle can be used to identify multiple (replicated) instances of a resource, thus distributing the load across many servers if required. 	<p>Both the browser and the server must be updated to support the handle system, and so resources can only migrate to Handle-specific servers (Sun and Lannom, 2000). In addition, all URLs must be changed to Handles for the system to work across the web, but Handles are not as human-readable as URLs. As such, although the Handle system can interoperate with the web, in order for it to become widely adopted, the Handle system must essentially replace the web.</p>

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach	Name	Description	Advantages	Disadvantages
Name Server (cont.)	Akamai	<p>Akamai is not specifically designed for resource migration, but enables resources to migrate and replicate in order to provide the load balancing service it offers. Akamai is a managed network that is separate from the Internet, with Akamai servers located at various ISPs across the world. The Akamai network will host resources on behalf of customers, providing a load balanced, fault tolerant service that replicates a resource and serves it to the client from the Akamai server closest to them.</p> <p>A customer must select the resources they wish to be managed by the Akamai network using software called the FreeFlow Launcher (Akamai, 1999). This replaces the hyperlinks in a HTML document with ones that link to an Akamai server. Thus, whenever a client requests an <i>Akamaiized</i> HTML document, it is served by the customer's server, but the links to other resources (such as images) embedded within it point to the Akamai network. As such, the Akamai network acts as a name resolution service, with its own name servers locating the resource that is closest to the user and serving it from its own Akamai servers.</p>	<ul style="list-style-type: none"> • Robust, scalable and provably optimal solution using state of the art algorithms for network load balancing. • Browsers do not need to be upgraded. • Retains the URL. • Deployed and operating service that has been successfully used by very large sites such as Yahoo, which serve over 100 million web pages per day. • Resources within the Akamai network can migrate and replicate very quickly, because the network is independent from the Internet. 	<p>The biggest disadvantage of the Akamai solution is that it is not a resource migration mechanism; rather, it enables resource migration as a side effect to its load balancing service. As such, only the replicated resources within the Akamai network can migrate, whereas the original resource on the customer's server can never migrate. The customer's server is therefore the weak link in the load balancing system, and the reason why Akamai cannot provide true web-based resource migration.</p> <p>In addition, resources can only migrate across Akamai servers, which are not even part of the Internet, let alone true web servers. Finally, the resource owner has no control over the resource migration, as the Akamai network determines where the resource should migrate to on behalf of the owner.</p>

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach	Name	Description	Advantages	Disadvantages
Callback	LinkGuard	<p>LinkGuard (Harris, 2000) is a commercial service that maintains a distributed database (called the LinkMap) of allegedly all URLs on the web, collected through customer submission and web crawling. When a resource migrates, software on the server originally hosting the resource informs the LinkMap, which updates its database and informs the servers of all other LinkGuard customers with links to that resource, so that their hyperlinks can be automatically updated. In this way, the LinkGuard system calls back servers with links to the migrated resource.</p>	<ul style="list-style-type: none"> • Optional system that users can choose to use if they wish. • No need to upgrade browsers. • Fully automated system allows information providers to forget about link management. • Retains the URL. • Creates a rich database of link information. 	<p>LinkGuard is a new system, and so although it has actually been deployed on the web, its success is unproven. Furthermore, it has a number of limitations:</p> <ul style="list-style-type: none"> • It is not an architectural solution. • It does not guarantee referential integrity, as only those web sites registered with the service receive a notification of a resource migration, and it cannot guarantee that all resources have been indexed. • All web servers involved, including those hosting the resource and all the hyperlinks pointing to it, must have the LinkGuard software installed. • The LinkMap will find it difficult to represent the real link structure of the web, as the largest search engine (Google) currently locates only 56% of the indexable web (Sullivan, 2000a). • The average age of a web page before its content changes (including any hyperlinks it contains) is just 117 days (Brewington and Cybenko, 2000), meaning that all resources in the LinkMap must be re-indexed every 117 days. • The LinkMap is a centralized database, whose scalability must therefore be questioned.

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach	Name	Description	Advantages	Disadvantages
Callback (cont.)	Atlas	<p>Atlas is a distributed hyperlink database system (Pitkow and Jones, 1996). An Atlas server partners a web server, and contains hyperlink information about that web server. This information extends the richness of the web's hyperlink by providing extended properties, such as the link's creator, its time of creation, and any permissions that are needed in order to use it. Each Atlas server knows all the outgoing links from each web page in the web server, and all incoming links. Whenever a resource migrates, the Atlas server checks its list of incoming links, and calls back those Atlas servers at the end of each incoming link, informing them that the resource has migrated. The hyperlinks contained within the Atlas server, and any relevant web pages, are then updated automatically.</p> <p>Atlas differs from LinkGuard by providing a distributed solution, with each Atlas server residing on or next to a web server, and informing all other relevant Atlas servers. In contrast, the LinkGuard's LinkMap is a centralized database.</p>	<ul style="list-style-type: none"> • Atlas provides all the benefits of the LinkGuard system, but using a distributed rather than a centralized approach. • Atlas is designed to be of use to an individual server, by providing local referential integrity, and in a distributed system of Atlas servers, by providing system wide referential integrity. • The extra information Atlas stores about the properties of the hyperlinks it manages enables it to offer other services, such as the ability to navigate backwards by letting the user follow incoming links. 	<p>Because of its similarity to the approach adopted by LinkGuard, Atlas suffers from many of the same problems, such as the need for a server to be upgraded in order for it to use the Atlas system. However, the Atlas system also suffers from the following problems:</p> <ul style="list-style-type: none"> • Poor scalability, particularly for large sites such as Yahoo, whose list of incoming links and outgoing links is enormous and extremely dynamic. In addition, it adds considerable network traffic, and will not scale to a system the size of the web. • Provides no mechanism for updating servers that are temporarily down. • Database of links for Yahoo would need to be almost as big as Yahoo itself, only more out of date, and would consume vast quantities of network capacity handling link requests and updates from external sites. • Particularly susceptible to Denial of Service attacks. • Alters the HTML contained within the resources.

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach	Name	Description	Advantages	Disadvantages
Callback (cont.)	Hyper-G (since renamed HyperWave)	<p>Hyper-G (Kappe, 1995) is another distributed hyperlink database, but rather than a node in the database communicating only with those nodes in the system at the end of an incoming link, the Hyper-G system uses the <i>p-flood</i> algorithm to flood the network and communicate with <i>all</i> nodes in the hyperlink database.</p> <p>Hyper-G essentially abstracts the link structure of the web from the web servers and places it in a separate and distinct distributed database. A resource migration is then treated as an event, which is broadcast throughout the system giving those servers with resources that have links to the migrated resource the chance to update those links.</p>	<ul style="list-style-type: none"> • No need to upgrade clients. • Fully automated system allows information providers to forget about link management. • Retains the URL. • Creates a rich database of link information. 	<p>Because of its use of a flooding algorithm, Hyper-G is completely unscalable to a system the size of the web. Hyper-G must store updates for servers that are powered off, and the number of messages that must be stored in a system with the size and unpredictability of the web would be completely overwhelming (Briscoe, 1997). In addition, despite the assertion that its network overhead is only 0.005% of web traffic (Kappe, 1995), this calculation still leads to a network overhead of some 150MB per day with only 1000 servers and 100,000 web resources (see Kappe (1995) for the calculation used). Today, there are over 3,000,000 servers (Lawrence and Giles, 1999), and over 1 billion web resources, which would generate an overwhelming network overhead of 450 Terabytes per day.</p>

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach	Name	Description	Advantages	Disadvantages
Chain	W3Objects	<p>W3Objects (Ingham et al., 1996) is an extensible web-based architecture, which objectifies web resources (which are then called <i>W3Objects</i>), and replaces the URL with a <i>W3Reference</i>. Whenever a W3Object migrates, a forward reference is left behind on the old server. If the W3Object moves again, another forward reference is used on the middle server. All three servers then communicate with one another, and 'short-cut' the chain by garbage collecting the middle W3Reference and ensuring the first W3Reference points to the server actually hosting the resource.</p> <p>Referential integrity is therefore guaranteed through strong link management on the servers themselves. In this way, the servers provide a distributed system of name resolution.</p>	<ul style="list-style-type: none"> • Solid, technically sound architecture, providing a complete, well managed, object-oriented architecture for the web and its resources. Provides a fast migration mechanism and robust garbage collection feature to ensure referential integrity on the web. • Essentially provides a web-specific distributed architecture, turning the web into a distributed platform. • Guarantees referential integrity. 	<p>The W3Objects system is not a web-wide solution, as it only works on servers that support the W3Objects architecture. Thus, a W3Object cannot migrate onto a non-W3Object server, limiting its use substantially. Worse, all browsers must be redesigned, or a proxy server used, in order for W3References to be used, and all 1 billion URLs must be replaced by the W3Reference if the system is to become a web standard. In addition, because the chain approach relies on the servers that host the resource to maintain the chain of references, it is a fundamentally fragile approach, and referential integrity would be lost if any server in the chain went down. Thus, the resource owner's reputation could be damaged due to the unreliability of someone else's server (see Evans et al. (1999)).</p>

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Approach		Name	Description	Advantages	Disadvantages
Search		Syntactic Clustering	<p>Syntactic clustering is a process used to determine the syntactic similarity of files and cluster those that are similar (Broder et al., 1997), but its use can be extended to provide a <i>Lost and Found</i> service that locates resources that have migrated (Broder et al., 1997). This is achieved by indexing all of the resources in the web over various periods of time, and maintaining discrete indexes at each point in time. It then locates a migrated resource by retrieving it from its index at a previous point in time before it moved, and then syntactically clustering all resources on the web at the current point in time that syntactically match it. The closest syntactic match should be the resource at its new location, provided it has migrated and not been deleted.</p>	<ul style="list-style-type: none"> • Can be provided as an external, third party service similar to a search engine, and so does not require any part of the web to be updated. • Can be used for many other services, as it provides a complete search service for any HTML document. • Locates syntactically similar (and therefore relevant) documents, so providing a search engine service as well. • Retains the URL. 	<p>The most obvious drawback of this approach is that it requires an index of every web resource both now and in the past. However, current search engines are struggling to maintain just one index of the web, without having to store multiple copies over time. Google (www.google.com) currently has the largest index, storing information on over 560 million (Sullivan, 2000) of the 1 billion (Inktomi, 1999) web resources available, but even this only indexes 56% of the web. As such, the system cannot guarantee referential integrity, as it may not be able to locate the resource. In addition, the resource would remain lost until the index was refreshed even if it managed to index all the web's resources.</p>

Table 1 - Analysis of Existing Migration Mechanisms (cont.)

Note that even though some of these examples are now over five years old, and the problem of link rot in distributed and hypertext systems has been known for decades, the web still has no effective resource migration mechanism, as the disadvantages presented in Table 1 have proved too great for successful adoption. The web therefore still suffers from link rot despite the danger it presents.

In addition, although each of the five approaches to resource migration provides its own solution to the problem of link rot, the semantics of the link and what it references are left in an ambiguous state. Referential integrity can ensure that links always reference the same resource, but what happens if the content contained within the resource changes? Should the semantics of the link insist that the content persists along with the resource, thus requiring a new resource and identifier to be created every time its content changes; or should the semantics be defined such that new content simply overwrites existing content? The former option will force web sites that contain frequently changing content, such as daily news sites, to use new resources with new identifiers every day, making external linking to the site virtually impossible, and leading to rapidly multiplying resources. Conversely, the latter option would only allow links to reference the web *site*, rather than a specific story on the site, requiring the user to manually search for the story within the site's archives (if they exist). In both cases, two separate resources have an equally valid claim to the same URL, but no resource migration mechanism for the web even recognizes this semantic ambiguity, let alone proposes a suitable solution.

Furthermore, existing migration mechanisms do not provide any support for *automatic* resource migration. They may provide name servers that can map a persistent identifier onto a varying location, but this must be done manually. Although this is irritating for the owner of a small site of perhaps one hundred web resources, particularly if the resources are distributed across several different servers, it renders the mechanism useless for large sites such as Yahoo's, with many millions of resources.

3.2.5 Summary of the Link Rot Problem

This sub-section has shown that link rot is a dangerous problem for the web, but existing solutions have failed to be adopted, as they have fundamentally ignored the web's philosophy, and the way in which its users use it. Link rot must be solved, but it must be through a solution that is sympathetic to and consistent with the web's current architecture, and which recognizes the unique way in which it is used.

3.3 Shrinking Namespace

The DNS has been in existence since 1985, but recently alarm has been raised at its shrinking namespace. Put simply, the number of domain names that are left unregistered is pitifully small, forcing the modification of the DNS itself in order to extend its namespace. Companies are suing one another over domain names for what they see as trademark infringement, while certain memorable domain names are commanding a premium of over \$1 million. However, the growth of the web, both in terms of new users and new web sites, is still exponential, and the number of people who wish to register a domain name will soon overtake the number of domain names remaining.

3.3.1 *The Cause of the Shrinking Namespace*

The DNS was originally designed to map a human-readable identifier onto an IP address in a distributed and scalable way. The actual domain names that were used did not matter, as it was only systems administrators and operators who used them, and so names such as *router1.rs-23.section7453.server12.east-gcb.sun.com* were common. In contrast, however, the URL, which includes the domain name in its syntax, has made the domain name far more visible, to the extent that it is now used in advertising and even in the brand name of companies. With the URL, the domain names do matter, as it is customers who must use them, and so there is a premium on memorable names, or those that represent the trademark of the company that owns the associated domain name.

In the rush for companies to be on the Internet, the domain name has become a symbol of a company's web presence, and appending the *.com* Top Level Domain name (TLD) onto a company's name associates that company with the web. The company *Amazon.com*, for example, explicitly includes in its company name the *.com* TLD that is part of the domain name of its server. This is because users generally type in the name of a company into a browser and wrap 'www'. and '.com' around it, and expect to locate the company's web site (indeed, this is what some browsers, such as Microsoft's Internet Explorer 5, do automatically). By appending *.com* onto their company name, therefore, the company implicitly associates itself with the web, while providing the user with the address of its web site in its brand.

This has subtly altered the semantics behind the domain name, as it must now identify a company or a product or a web site, and not just a server. Originally, the domain name was a simple mapping from a human name to a machine name; with the advent of the web, however, the domain name is now an identity, which has led to the dramatic shrinking of the *desirable* namespace (i.e. the space of all names that are wanted and will be used, as opposed to names comprising arbitrary characters, which may be syntactically legal, but which will never be used).

This subtle shift in the semantics of the domain name has also altered the semantics behind the operation of the DNS, effectively turning it into a rudimentary directory system (Mitchell et al., 1996). For example, users will append '.com' to the name of a company, or '.edu' or '.ac.uk' (depending on the geographical location) around a university's name, in order to locate the respective organization's web site. In this way, the users are implicitly using the TLDs of the DNS as the top level of a hierarchical directory structure. The problem is that the DNS was never designed to be a directory system, and a number of problems, both technical and social, have now begun to emerge.

3.3.2 *The Damaging Effects of the Shrinking Namespace*

The shrinking namespace brings with it problems that are both technical and social. The technical problems are derived from the way in which the semantics of the DNS are being altered to turn it into a system it was never designed to be, without its architecture changing to adapt to this shift. The social problems, on the other hand, derive from the fact that the DNS namespace is global in presence but limited in size, providing fertile ground for conflict as a

limited resource is suddenly made valuable. The most damaging effects of these problems include:

- *Social, political and legal tension*

The namespace of the DNS is essentially flat, as everyone wants to use *.com*, and a domain name must be globally unique. However, company names are not unique (Mitchell et al., 1996), even in the same country. This leads to tension and ultimately litigious conflict over who owns a specific domain name (for example: who owns the domain name *mcdonalds.com*? The giant hamburger chain or the local baker who has been in business 50 years longer?), which in turn brings trademark law into the dispute. However, trademark law is itself contentious, and inconsistent across different countries.

- *Hyper-inflated domain name prices*

The combination of the huge demand for domain names as the web continues growing, and the drastic shrinking of the desirable namespace, has led to the price of domain names reaching hyper-inflated levels, with simple, easily recognizable names such as *Drugs.com* being auctioned for over \$1,000,000 (Arent, 1999). This will cause the namespace of the web to fragment into two classes: those who can afford a desirable domain name, and the unlucky majority who cannot, a situation that is the antithesis of the web's philosophy.

- *A damaged DNS* – because of the pressure on the *.com* namespace, and the demand for a memorable name, international domain names are now being used to provide memorable names that use an international TLD for purposes other than identifying a server's country

of origin. For example, the small island of Tuvalu recently sold its *.tv* international domain name to the company DotTV, which then registered *.tv* domain names, such as *bbc.tv*, in the hope of selling them to TV companies for vast sums of money. Equally, Chung Minh Shih uses Armenia's *.am* international domain name to provide memorable names such as `http://i.am/john` (Oakes, 1998). However, this erodes the semantics of the DNS (Oakes, 1998) and blurs its functional definition, changing it from a simple hostname/IP address mapping service, into a directory system in the case of *.tv*, and a membership system in the case of *.am*. The problem is that the DNS was only ever designed to be a simple hostname/IP address mapping service, and all other uses for it place unknown demands on its ill-prepared architecture.

3.3.3 *Determining the Extent of the Problem*

How close is the namespace of the DNS to exhaustion? The theoretical limit of the namespace can be calculated using the figures given in the DNS specification (see Mockapetris, 1987b). A domain name string can contain a maximum of 255 characters, with each character being selected from a pool of 28 different types (26 letters of the alphabet (a domain name is case-insensitive) plus the characters '.' and '-'). This puts the number of unique names in the DNS's namespace at:

$$((26+1+1)*255) ! = 7.3888253549170121004175301528e+24416$$

However, although this is a truly vast number, infinite scaling of the DNS is technically unworkable (Mitchell et al., 1996), and so this limit will never be reached. In addition, the

DNS will only be required to scale to the limit of its *desirable* names, not its theoretical limit, and so the number of desirable names remaining is a better indicator of the size of the DNS's remaining pool of domain names.

There are currently 31,050,574 domain names registered (see DomainStats (2000) for a continually updated figure), and as Figure 2 shows, the number of domain name registrations is increasing exponentially, with the number expected to reach more than 75 million by the end of 2002 (Barrett, 2000). Clearly this does not even scratch the surface of the theoretical limit of the namespace, but what about the desirable limit?

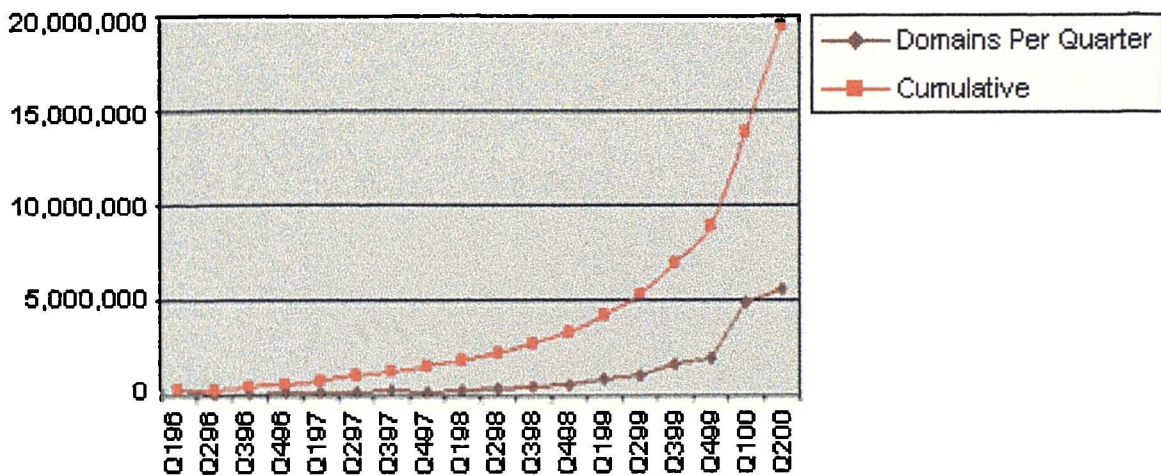


Figure 2 - Total number of domain names registered by quarter (DotCom, 2000)

The DNS currently has 7 TLDs:

- *.com* – for commercial organizations;
- *.net* – for networking organizations, like NSFNET;
- *.mil* – for military organizations;
- *.edu* – for educational organizations;
- *.org* – for noncommercial organizations, like the IETF;
- *.gov* – for government organizations;
- *.int* – for international organizations, like NATO.

There are also international TLDs, such as *.uk*, *.au*, *.de*, etc., which represent the various countries around the world. However, as Figure 3 shows, 80% of all domain names use the *.com* TLD. This shows that nearly all other TLDs (and with them those parts of the total DNS namespace which those TLDs represents) are perceived as undesirable, making them effectively unusable, and reducing the structure of the overall namespace down from a hierarchical namespace to a flat one instead.

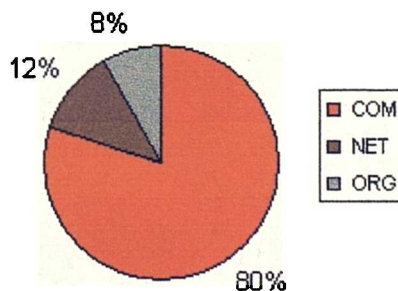


Figure 3 - Percentage of registered domain names according to TLD (DotCom, 2000)

Worse, of those *.com* names that are perceived as usable, most of the best (i.e. the most memorable or descriptive) are already gone (Arent, 1999), with a Wired News investigation conducted in April 1999 finding that out of 25,500 standard dictionary words checked, only 1,760 remained unregistered (McCullagh, 1999). It is safe to assume that since then, the vast majority of those 1,760 will by now also be registered.

The remaining desirable domain names are either not in the dictionary, or are hyphenated constructs of more than one word (which is prone to error when being typed into a browser's address bar). However, if the definition of *useable* is restricted to company names, or single words in a standard dictionary, then it can be seen that virtually all of the useable domain names have already been registered, at least for the *.com* namespace. In this way, the namespace of the DNS has been reduced from a vast hierarchy of names to a flat, almost exhausted pool of unwanted, meaningless names. It is the limit of the useable namespace that causes the most problems for the DNS, and this limit has very nearly been reached.

3.3.4 Solutions to the Shrinking Namespace Problem

The problems faced by the DNS are the direct result of it being adapted to fit functions it was never designed to perform. For example, the DNS is being asked to provide a naming system, a directory system, and even a company's brand identity on the web, but each of these different functions place different and conflicting demands on the system.

Other name resolution systems are more focused in their operation, and are designed to perform one function, and to perform it well. Table 2 on page 83 provides a comprehensive

list of the different types of naming systems that are in operation today, and describes their function and the environment in which they are designed to operate, including whether they are designed for human use or machine use. Note how the functionality of each system is markedly more focused than that of the DNS, which has evolved to perform many different functions, and to be used by users with many different levels of experience.

3.3.4.1 The Irreplaceable DNS

Of the different naming systems described in Table 3, those designed for distributed component architectures such as CORBA are designed for machine use only, whereas those designed for network file systems are little different in operation than the DNS, and so would suffer the same problems. As such, only the directory service provides a realistic alternative to the DNS. Indeed, the OSI X.500 reference model for directory services, and its more lightweight derivatives such as the Lightweight Directory Access Protocol (LDAP) (Yeong et al., 1995), and Novell's Network Directory Services (NDS), have been seen as *competitors* to the DNS (Albitz and Liu, 1997). They provide a globally consistent, hierarchical namespace that is used to locate a resource (UniOfMich, 1995), but which can be extended to reference a resource of any type, allowing resources to be viewed consistently, no matter what their object type. In addition, although they are relatively slow when adding or deleting users and resources, they can be updated securely across the network (Albitz and Liu, 1997). As such, they provide a far richer naming system than the DNS.

Name Resolution System	Human or Machine Readable Name	User Level Expected	Purpose	Description
Local File system	Human Readable	Reasonably proficient	Identifies a file in a file system according to its location.	A computer's local file system has its own namespace, with its resources being files that are named according to the conventions of the operating system, and the naming system being a component of the operating system. The file system must ensure the uniqueness of each file name; provide functions for adding, deleting, creating and renaming files; and associate a name with a specific block of data. The namespace is operating system-dependent, and usually encodes the file name and file location in a single filepath, as a human-readable string.
Network File system	Human Readable	Reasonably proficient	Identifies a file across a network according to its location	With the introduction of networks, the functionality of the naming system must grow to accommodate the extra complexity introduced by the distributed nature of the system. Such a naming system must cope with resources across potentially thousands of machines, ensuring the uniqueness of the name, locating each individual resource unambiguously, and providing added services depending upon the type of distributed system. Microsoft's Windows Internet Name Service (Microsoft, 2000a), for example, provides a distributed database that maps Windows-specific computer names to an IP address, and whose namespace encodes the computer name as well as the filepath, as a human-readable string. In this way, WINS, as it is known, is similar in functionality to the Internet's DNS, but provides a Windows-specific service whose records can be dynamically updated across the network should the machine be assigned a different IP address

Table 2 - Name Systems in Use Today (continued on following pages)

Name Resolution System	Human or Machine Readable Name	User Level Expected	Purpose	Description
CORBA Name Resolution System	Machine-readable	Expert	Names a CORBA object across a CORBA distributed component system	<p>The Common Object Request Broker Architecture (CORBA) is an object-oriented distributed architecture designed by the Object Management Group. CORBA's defined naming system is the CORBA Naming Service, which provides the principal mechanism through which most clients locate computational objects that they intend to use (OMG, 2000). The CORBA Naming Service uses several <i>naming contexts</i> in a hierarchical system to fully resolve an object's name onto its address.</p> <p>Interestingly, CORBA's namespace does not use a specific syntax. Rather, a CORBA name comprises an <i>identifier</i> attribute, to identify the object within the system, and a <i>kind</i> attribute. The latter provides descriptive power to the name, as CORBA does not interpret, manage or even attempt to understand the syntax used. This is left to higher levels of software, which can impose their own management policies on the naming of objects (OMG, 2000). In this way, CORBA has its own namespace for its own uses, but also provides an unrestricted namespace that can be organized according to the requirements of the higher layers of software that use the CORBA system.</p> <p>As well as having a more sophisticated namespace, CORBA, as a platform for distributed computing, also has a more sophisticated naming system. For example, an object's <i>externally visible characteristics</i> (OMG, 2000), such as its read/write attributes, last-time-modified attribute, or other properties, can be registered with other CORBA services and used to locate the object. In this way, the name service can be used in conjunction with the CORBA Query Service, for example, to enable clients to search for an object according to a query.</p>

Table 2- Name Systems in Use Today (cont.)

Name Resolution System	Human or Machine Readable Name	User Level Expected	Purpose	Description
Directory Services	Human-readable	Reasonably proficient	Treats everything accessible from the network (including a computer, a fax, a printer, etc.) as an object in the directory, and enables it to be located, used, queried, and managed in a consistent, homogenous way.	<p>A directory service is a database that manages the attributes and locations of shared objects across a network, where objects include anything from a printer to a file to a user (Esposito, 1999). The directory's hierarchical namespace is used to locate an object, which is classified according to a specific category in the directory that is reflected in the object's name.</p> <p>A directory service is like a phone book, as it provides information about a person or a resource when given their name (Microsoft, 1997a). It combines multiple directories, such as file systems, email contact names, or those from different groupware products, into one consistent place, and provides advanced security features to ensure that each object is accessed only by those with appropriate authorization.</p> <p>Directory services are designed to provide regular queries, but few updates, and so are relatively slow when adding or deleting users and resources, but they can be updated across the network in a secure fashion (Albitz and Liu, 1997).</p>

Table 2- Name Systems in Use Today (cont.)

Name Resolution System	Human or Machine Readable Name	User Level Expected	Purpose	Description
Domain Name System	Human readable	All levels, from expert systems operator, to reasonably proficient Internet user, to complete novice.	Originally designed to map a human-readable server name onto its IP address, but is now used to identify: <ul style="list-style-type: none"> • any server on the Internet; • a web site according to a directory structure; • a company brand name or trade mark; • a product; • a person. 	<p>The DNS was originally designed to map a human-readable hostname onto a machine-readable 32-bit IP address. It provides a hierarchical namespace that is used to locate the server's IP address. However, a domain name is now also used in a variety of different situations, each with different requirements of the DNS.</p> <p>The DNS is a read only database, which cannot be updated across the network. Its security is questionable and its features limited, but it is an integral part of the Internet that is simple, mature, and robust, and will be extremely difficult to replace.</p>

Table 2- Name Systems in Use Today (cont.)

However, the DNS is integral to the Internet, not just the web, and is a fast, simple, and robust system that is now mature and extremely reliable (Albitz and Liu, 1997). Replacing a name resolution system that is used by over 369 million people (GlobalReach, 2000) with something as fundamentally different in operation as a directory service is impossible, as it would require stopping the entire Internet while the new system is integrated and tested with every different protocol and application that relies on the DNS. As such, although the semantics of the DNS have become blurred, it can still demonstrably cope with enormous numbers of users and a vast array of applications without fail. In contrast, no directory service has had to scale beyond the enterprise, and so has only been tested within a more controlled environment. As

such, even a directory service may find its semantics blurred when placed in the service of the users of the web.

In addition, replacing the DNS with a directory service implicitly assumes that the web's naming service *should* be organized as a directory, but this requires centralized control in order to organize the directory structure. Worse, directories such as X.500 are unwieldy, both for the user, as the names used are much too long for normal use, and for the machine, as X.500 requires far more powerful computers than the DNS, with standard PCs only able to use its lightweight variants such as LDAP and NDS (Mitchell et al., 1996).

3.3.4.2 The Inextensible DNS

If the DNS cannot be replaced, then it is reasonable to expect it to be extended. The body responsible for managing the DNS's namespace is ICANN, the Internet Corporation for Assigned Names and Numbers, and it has proposed a number of possible extensions in an attempt to adapt the DNS to its new web-oriented environment. In order to settle disputes over contentious domain names, ICANN has proposed the Uniform Domain Name Dispute Resolution Policy (ICANN, 1999), a formal arbitration process through which conflicting parties can argue their case before suing one another in court. More positively, perhaps, ICANN has also endeavoured to open the namespace up by providing more TLDs that are domain-specific.

.ads	.find	.mus
.africa	.firm	.nom
.air	.geo	.one
.biz,	.health	.per
.cash	.i	.pid
.co-op	.info	.post
.dir	.jina	.pro
.dot	.kids	.tel
.dubai	.law	.travel
.event	.mall	.union
.fin	.mas	.web

Table 3 - New TLDs submitted to ICANN

Table 3 presents a list of some of the new names that were presented to ICANN as potential TLDs. However, after much debate, the organization eventually settled for just seven (Table 4).

.aero	.museum
.biz	.name
.coop	.pro
.info	

Table 4 - New TLDs chosen by ICANN

ICANN proposes that the new TLDs will provide many more desirable domain names, and so solve the problem of the diminishing namespace. However, this approach is fundamentally flawed. Although there may be seven more TLDs, and a corresponding increase in the number of domain names that can be registered, it will not be long before the same disputes occur for domain names under the new TLDs. For example, *.biz* is meant to represent the namespace for e-commerce, but it is not clear how it will suffer any less than the *.com* namespace, with, for example, the UK's Dixons fighting with the US's Dixons for the right to the *dixons.biz* domain name. In addition, the different mix of domain names all place different semantics on the DNS. For example, domain-specific TLDs, such as *.museum*, choose to assume that the DNS is a directory service, and the TLDs are categories within the directory; other TLDs, such as *.info* treat the DNS as a service locator, with the TLDs being used to define different types of service. Still other TLDs, such as *.name*, treat the DNS as a directory of people, while it is left unexplained why there is a *.pro* but no *.amateur*, or why there is a *.museum* but no *.gallery*. The mix of different types of TLD is eclectic at best, but they are all at the same level in the hierarchy. No other directory would place international country codes, such as *.uk*, at the same level in its hierarchy as business categories, services, and people, and it is difficult to see exactly why the DNS should. Indeed, the mix of new TLDs is so arbitrary, it would be difficult to define the semantics of the DNS at all.

Opening up the namespace in this way does not solve the problem, it merely delays its full effect. The fact is that with a centralized body such as ICANN controlling the namespace, there will always be a restricted name space, and therefore high demand for certain key names. Unfortunately, ICANN does not seem to be helping the situation with its arbitrary mix of new

TLDs. As such, the DNS can technically be extended, but until its exact functional definition is determined, it seems any extension will not solve its problems.

3.3.5 Summary of the Shrinking Namespace Problem

This sub-section has shown that the DNS is facing a crisis, with the number of desirable domain names left unregistered reduced to virtually zero, forcing users to employ cunning workarounds that make a domain name more memorable, but which undermine the semantics and operation of the DNS. The namespace must be extended to stop the development of a stratified web, divided according to money, but care must be taken to decide firmly on the exact semantics that should underlie the DNS. It must be determined whether or not the DNS should retain its role as a database for resolving host names and IP addresses, or whether it should become more like a directory service. If the latter is chosen, ICANN would argue that it should hold the responsibility for maintaining and defining such a directory, but its solution to the shrinking namespace problem is fundamentally flawed and ill thought through. ICANN's problem is that it is one of the only centralized bodies on the Internet, and as such, cannot hope to provide a global solution that meets everybody's requirements. However, the problems with the DNS must be solved quickly, and in a way that is decentralized, and which reflects the needs of the web and its users.

3.4 Increasing Noise

The architecture of the web has no explicit mechanism for managing its information, and has relied instead on ad hoc services provided by third party service providers, who respond to the demands of the market. However, this approach is failing, as search engines are indexing less of the web and returning less relevant results, while hyperlinks are becoming completely

unreliable. As the web grows in size, the quality of its information degrades, leading to an increasingly noisy system.

3.4.1 *The Cause of the Increasing Noise*

The root cause of the problem is the volume of information combined with the web's decentralized architecture and lack of proper information management. Lagoze and Fielding (1998) define the problem well, reducing it down to three components:

- *Universality* – anyone can participate equally on the web, leading to a system that inherently focuses on quantity over quality. This is inevitable on the web, as quality requires the classification of one item of information as being better than another, but without a gatekeeper, the web has no mechanism with which to do this. As such, all information is treated the same, regardless of its source or its authority. It is left to search engines to attempt to infer the quality of information using heuristics that analyse the text of an HTML document.
- *Uniformity* – resources, services and users are treated as equal, when they clearly are not. Indexing image content is distinctly different from indexing text, and users clearly have different levels of experience, yet the web makes no distinction.
- *Decentralization* – the organizational structure required to manage the information effectively cannot be put in place on the web, leading to an anarchic structure that gets more extreme as the web grows.

Put simply, there is no efficient information retrieval system inherent within the web's architecture that users of different abilities can use to retrieve information of different media types. This problem is compounded by the decentralized nature of the architecture, which prevents the development of such a system, and the web's phenomenal growth, which constantly exacerbates the problem.

Compounding the problem further is the state of the web's navigational mechanism: its hyperlinks. Section 3.2 has already discussed the problems due to link rot, but the effectiveness of the hyperlink is also compromised through hyperlinks that deliberately misdirect the user, or which pay for their location on a web page, regardless of their relevance to its content. These and other tactics lead to the breakdown of the web's navigational structure, which is already in a fragile condition. The end result is that both the search engine and the hyperlink are becoming increasingly unreliable, therefore making the web increasingly un-navigable.

3.4.2 The Damaging Effects of Noise

Noise is defined as unwanted signals, and accompanies any data transmission event (Stallings, 1991). In an information system such as the web, noise is represented by unwanted information content. In the specific case of a search engine, for example, if one web page out of one hundred is perceived as relevant by the user, the other 99 irrelevant web pages represent noise. The more noise in such a system, the harder it is to locate information.

At its most extreme, noise in an information system will kill it, as the number of unwanted signals will grossly outweigh the number of wanted signals, thus rendering the system useless. Noise represents the quantity of entropy present in a system, which is a measure of the randomness or unpredictability of communicated values (Brebner, 1997). Maximal entropy represents a completely random and thus uniform system, making entropy the polar opposite of information. In this way, increasing noise renders an information system increasingly random, and therefore poses a potentially lethal threat to the web.

However, even at low levels it can severely retard the growth of the web, as new users find themselves overwhelmed with information they do not want. As was said in section 2.4.2, without any system-wide structure to the web's information, more people will perceive each new web page as noise than those who perceive it as relevant information. New users, whose inexperience will cause them to use search engines and hyperlinks ineffectively anyway, are particularly susceptible to this problem, which will act to implicitly raise barriers to adoption and retard the web's growth.

3.4.3 Determining the Extent of the Problem

There are two ways that a resource can be located on the web: through hyperlinks, which connect related documents; or through an information retrieval service such as a search engine. As such, if these mechanisms are ineffective in locating relevant information, the user will perceive the web as a noisy system. The following sub-sections will therefore attempt to determine the level of perceived noise in the web through an analysis of the literature relating to the state of these two different navigational mechanisms.

3.4.3.1 *The State of Hyperlink Navigation*

3.4.3.1.1 *Navigation Mechanisms*

The user interface of a web browser provides navigational features that interact with the web in order to help the user navigate across it. In a study examining the revisitation patterns in web navigation, Tauscher and Greenberg (1997) classified the following as major navigation features of a browser:

- *open URL* – the user types a URL into the browser's address bar;
- *back* – the user hits the browser's Back button, to return to the previously viewed resource;
- *reload* – the user reloads the current page from the server;
- *forms* – the user submits a form via HTTP, using a button in the HTML document.
The returned resource is usually dynamically generated;

In addition, Catledge and Pitkow (1995), in their study into browser characterizations, also included:

- *forward* - the user hits the forward button, revisiting a page they have just come from;
- *home* – the user hits the Home button to load a resource which they have pre-selected as their ‘home-page’ (i.e. the default page that is loaded when the browser is first started);
- *history* – a list of all the URLs visited in a pre-defined time period is presented to the user, who then selects one to navigate to;

Finally, the user can choose to explicitly click on a rendered hyperlink, or select from a list of URLs that have been stored by the user in her *Favourites* list. Different browsers may adopt other navigational features, but these constitute the main ones that are common across all modern browsers. The most commonly used features have been found to be the hyperlink (51.9%) and the Back button (40.6%) (Catledge and Pitkow, 1995).

A user must combine these navigational features of the browser with effective search heuristics, if they are to successfully navigate across the web without the aid of a search engine. For example, Tauscher and Greenberg (1997) found that users visit a central page and navigate to and from its many linked resources, thus performing a breadth-first search. Kleinberg (1998) has termed these central pages *hubs*, while the linked resources are

authorities, which satisfy a user's information need and are perceived by that user as being authoritative and therefore accurate. Tauscher and Greenberg (1997) also note that users may follow a 'guided tour', composed of hyperlinks containing instructions such as 'Next Page' that are followed by most users according to a set structure; or they may perform a depth first search, following hyperlinks deeply before returning to a central page. Higgins (1999) has also noted that time and authority affect human decision making, with humans deciding which item of information to choose according to the authority associated with each item, and the time available to decide. From this, it can be seen that the effectiveness of hyperlink navigation is dependent upon the navigational features of the browser, the information contained within the web resource, and the user's own search heuristics.

3.4.3.1.2 *The Problem with Browsing*

Unfortunately for the user, neither the browser nor the information contained within a web resource is particularly helpful in locating a specific resource. Cockburn and Greenberg (1999) note the following limitations of the browser's navigational features:

- *Inefficiency in retrieving distant pages* – the Back button only works one page at a time, which is a laborious process when the user has visited many pages.
- *Context* – the user sees only one web page at a time, and so their orientation within the information space is dependent upon the contents of the current page and their memory of any previous page.

- *URLs do not make good lists* – favourites and history mechanisms that list URLs are not intuitive, as most URLs are not representative of the content of the resource that they locate.

In addition, with many links on every page, it is easy for the user to become distracted (indeed, web-based banner advertisements depend on distracting the user), and to then forget where in the browsing session they were. Furthermore, the navigational cues in the browser only allow the user to see hyperlinks that are one level deep. There is no way to see what the resource that the hyperlink points to is, or of knowing what hyperlinks may be contained within it, without clicking on the hyperlink. This severely slows down the user's browsing progress, and renders browsing an almost arbitrary approach to information retrieval for all but the most experienced of web users. Worse, hyperlink navigation can only be as good as the state of the hyperlink structure itself, but as the following sub-section shows, this is disintegrating at an alarming rate.

3.4.3.1.3 The State of the Web's Hyperlink Structure

The structure provided by the web's hyperlinks is the only system-wide form of information management inherent within its architecture. Users can theoretically use hyperlinks to navigate from one resource to any another. Albert, Jeong, and Barabasi (1999) have shown that any two randomly chosen documents are, on average, only 19 hyperlinks away from one another, which, when combined with the use of advanced navigational techniques, theoretically allows the user to navigate across the web using hyperlinks alone. However, in practice this is not the case, as a much larger study of over 200 million documents by Broder

et al. (2000) has shown that the web has a complex, organic structure, with only a 24% probability that any two documents are connected via hyperlinks at all, and that the actual diameter of the web (that is, the number of hyperlinks that must be traversed between randomly chosen documents) is closer to 500. Figure 4 shows this structure, which reveals that the web has a rich inner core of some 56 million highly connected resources that connect to and from one another. However, there are also 44 million IN resources (i.e. those that link to the core, but which cannot be reached from the core) and 44 million OUT resources (i.e. those that are linked by the core, but which do not link back to the core). Worse, Broder et al. (2000) also found that there are some 44 million resources that bypass the central core altogether, and another 17 million pages that are completely disconnected.

This structure shows that the web cannot be completely navigated using hyperlinks alone, and so the web's inherent navigation mechanism cannot be used in isolation. The situation is made much worse, however, by hyperlinks that pretend their referenced resource relates to something that it clearly does not. This deception is intended to attract as much user traffic to the resource as possible, regardless of whether or not each user actually wants the information it contains. However, it erodes the integrity of the hyperlink structure of the web, further increasing the noise level. As such, the hyperlink can no longer be relied upon for effective navigation.

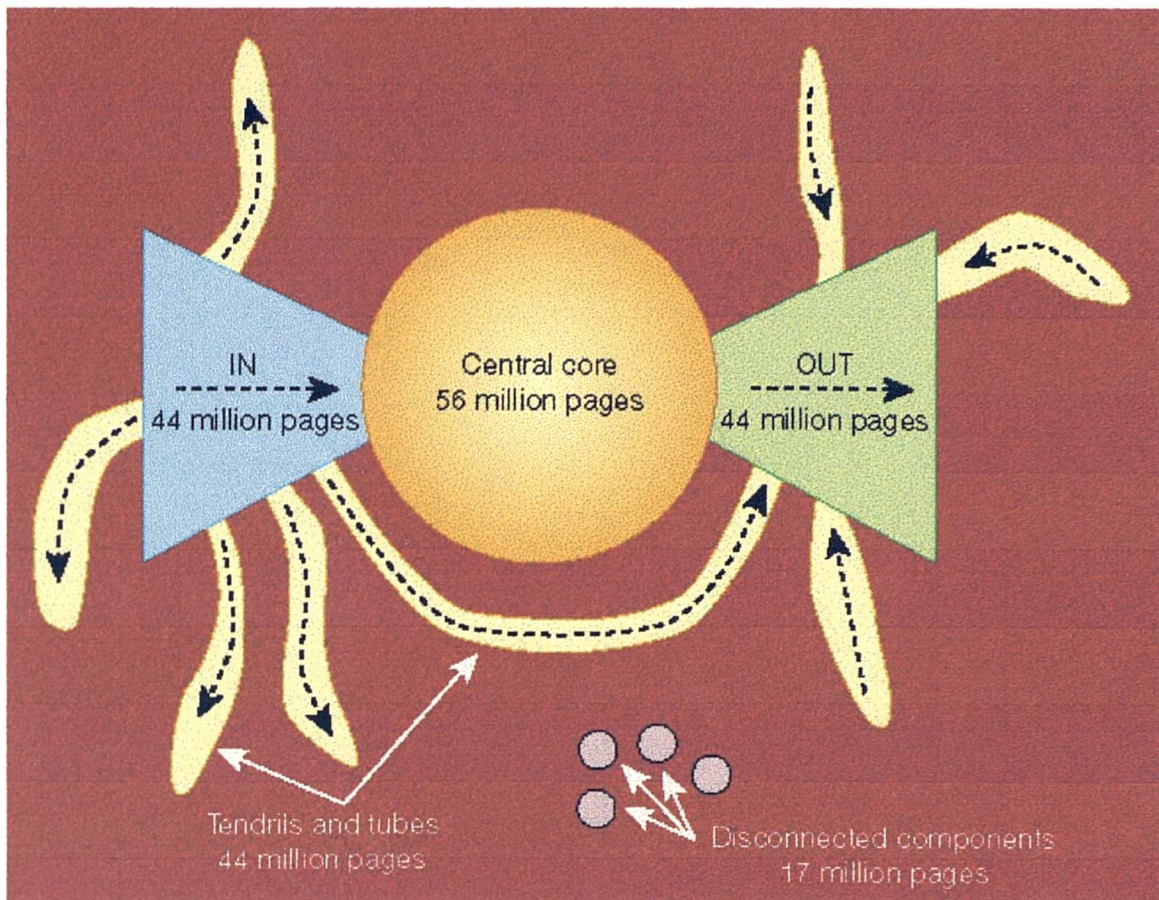


Figure 4 - Hyperlink structure of the web (Broder et al., 2000)

3.4.3.2 The State of the Web's Information Retrieval Services

Third party information retrieval services are deployed on top of the web's architecture, but are not part of it. As such, they must index that part of the web that they wish to focus on without help from the web's architecture, but they are free to choose whichever mechanism or algorithm they wish in order to achieve this.

The web's main information retrieval services can be classified according to two categories:

- *Coverage-oriented services* – services, such as the major search engines, that try to cover as much of the web as possible.
- *Relevance-oriented service* – services, such as web directories, that focus more on providing relevant results than on attempting to index the whole web.

The following sub-sections examine the state of the services that belong to these categories.

3.4.3.2.1 *Coverage-Oriented Services*

Search engines attempt to index the entire web using web spiders, and must infer the meaning of a document using machine-based heuristics. However, search engines are facing three critical problems:

1. *Web crawling is no longer viable*

It is becoming increasingly difficult for a web spider to keep up with the growth of the web. Of the web's one billion resources, today's largest search engine (Google) indexes only 56% of them (Sullivan, 2000a). However, the situation can only get worse, as the web is expected to hold some 100 billion documents by the end of 2002 (Butler, 2000), while the web's hyperlink structure, which the spiders rely on to efficiently locate new documents, is fragmenting. With only 24% of web resources connected at all, crawling hyperlinks is no longer viable if the goal is to index all web resources.

2. *Relevance is based on unreliable inference heuristics*

Search engines must attempt to infer the meaning of a document in order to return relevant results, yet they generally use outdated relevance heuristics (Berst, 1998), and choose instead to compete on the size of their index rather than its accuracy. This leads to what is known as the *abundance problem*, in which the number of resources classified as relevant by the search engine is far too large for a human to digest (Kleinberg, 1998). For example, generic search terms such as 'web' can yield as many as 250 million returned documents, greatly increasing the perceived amount of noise in the service.

3. *Indexes age quickly*

The average age of a web page before its content (including any hyperlinks it contains) changes is just 117 days (Brewington and Cybenko, 2000), meaning that every document indexed in a search engine must be re-indexed within 117 days if the index is to remain fresh. However, with the size of the web increasing exponentially, this means that the number of documents that must be refreshed must also increase exponentially. The problem is compounded by link rot, which has already been covered in section 3.2.

Feldman (1998) has provided empirical data on the noise level of a search engine, by conducting a study among 999 professional information searchers that compared the difference between the Dialog controlled information service with the web's search engines. In this study, clients of professional searchers used real world queries, and were asked to rate

the relevance of the returned information. The results, shown in Figure 5, reveal that although both services return nearly the same amount of highly relevant documents (111 for the web against 117 for Dialog), the web returns nearly twice as many irrelevant documents (306 to 147).

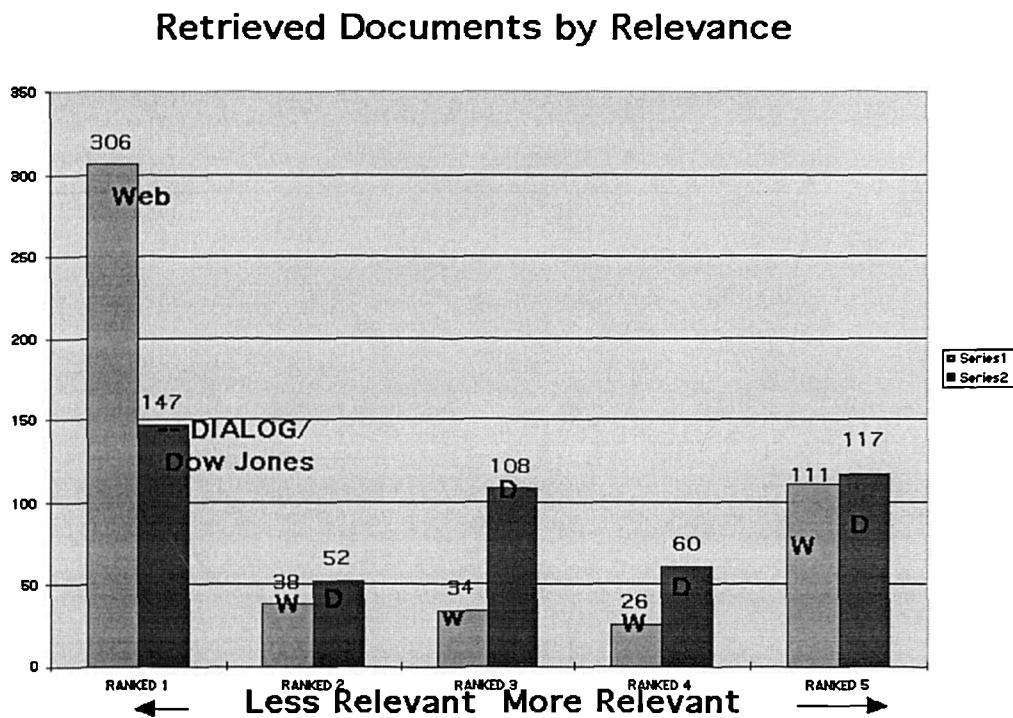


Figure 5 - Relevance of the Web Compared to Dialog (Feldman, 1998)

Worse, of those documents that rated a relevance score of 4 out of 5, the web returned only 43% as many as Dialog (26 to 60), while for those documents with a relevance score of 3 out of 5, it returned only 31.5% (34 to 108). Feldman (1998) notes that:

“...the interspersal of so many useless documents with those of high value may colour the perception of the searcher that the entire Web search has less value than a traditional online search, even though the same number of highly relevant documents were returned.”

Thus, regardless of the number of relevant documents that are actually retrieved, the user's perception is that the search engine is noisy. Worse, Henziger et al. (1999) note that the more pages an index contains, the harder it is to keep the average page quality (in terms of relevance) high, which means that the efforts of the search engines to provide the largest index inherently lowers their average page quality. As the size of the web continues to increase, the average page quality, and thus the quality of the search engine itself, will get progressively worse.

These problems terminally undermine the approach of the search engine companies. They cannot index the entire web; their systems are increasingly noisy; and their indexes are becoming increasingly stale. As the web's only comprehensive information retrieval systems, they only add to the perception of increasing noise in the web.

3.4.3.2.2 *Relevance-Oriented Services*

Other information retrieval services, such as human-indexed directories, focus more on relevance than coverage. The directories, for example, attempt to index the web by hand using thousands of editors. However, the size of the resultant directory is considerably diminished, with the largest directory, the Open Directory (www.dmoz.org), currently indexing only 2,000,000 documents (Sullivan, 2000a), compared with Google's 500,000,000. Worse, with so many documents to classify, the directory structure can become unwieldy, forcing the directory operator to decide whether to use a small directory structure, with each level containing millions of different resources, or to limit the number of resources per directory level, with millions of different directory levels.

Other types of service include domain-specific search engines that focus on one specific subject (termed *vortals*, for vertically-oriented portals). These services, however, cover even less of the web, as they ignore all resources not related to their specific subject in an attempt to increase relevance. As such, the relevance-oriented approach cannot provide a comprehensive information retrieval service, and so although it may have its use for certain groups of users, the approach cannot reduce the overall noise on the web.

3.4.3.3 Implicit Gatekeeping

Ironically, a relevance-oriented service cannot become a core part of the web, as the service itself conflicts with the web's core philosophy that has made it so popular. Specifically, the service becomes a gatekeeper, deliberately choosing one item of information over another. This is self-evident, as the concept of relevance demands such discrimination.

However, all services have their own bias regardless of their orientation, and so become gatekeepers, even if only indirectly. This may be an explicit bias, particularly with the coverage-oriented services, whose editors must follow the editorial line of the service provider; or an implicit bias, more common with the relevance-oriented services, whose machine-oriented heuristics inevitably judge the relevance of a resource according to features other than those directly describing its information (Lawrence and Giles, 1999). For example, Google's use of the PageRank algorithm (Brin and Page, 1998) leads it to rank documents according to their popularity, regardless of the informational content contained within them. This has led some unscrupulous site operators to deploy thousands of entry sites, which contain no content other than a distinct hyperlink to the same web site, in an attempt to

artificially inflate their PageRank score. Worse still is the practice employed by some search engines (such as *goto.com*) of actually *selling* rankings to the highest bidder, regardless of their relevance to a query (Berst, 1998).

The cause of implicit gatekeeping is the lack of an architectural solution to information retrieval. Search engine companies exist in a fiercely competitive world, where the vast majority of their services are provided for free. This puts enormous pressure on them to make money from any available source, and to take money away from their search technologies and give it to their marketing departments instead (Berst, 1998). In this way, they lose their coverage, their relevance and their neutrality, and provide an ineffective service to the user.

3.4.4 Summary of the Increasing Noise Problem

This sub-section has shown that increasing noise in the web is a dangerous problem that is being exacerbated by the tactics of information service providers. The web's hyperlink structure is being eroded by hyperlinks that deceive the user in order to generate traffic, or by web sites that do not provide any hyperlinks at all to competing, but relevant web sites. Equally, search engines are now selling high-ranking scores regardless of a site's true relevance. However, worst of all is the fact that there is now no way to navigate the entire web using web-based services alone. Hyperlink navigation can no longer be relied upon, as the hyperlink structure has broken down, leaving some resources completely disconnected. Equally, information retrieval services can no longer be relied upon, as they cannot keep up with the growth of the web.

In short, the web is now perceived as a noisy system. This will discourage users from using it effectively, and will begin to retard the web's growth. However, if left unchecked, the noise will increase and will eventually render the web useless. A new way of enabling users to locate resources effectively is required, which should be completely unbiased, comprehensive, and tailored according to the needs of the user and the information provider.

3.5 *Summary*

The chapter has focused on three core problems faced by the web:

- Link rot
- Shrinking Namespace
- Increasing Noise

The problems have been recognized for some time, and various proposed solutions have been described in this chapter. However, they have all failed, and the web is left with a flawed architecture that threatens its growth and even its existence.

The problem with the existing solutions has been that they are unsympathetic to the architecture of the web; the needs and behaviour of the users; and the needs and behaviour of the information providers. Specifically, a system that requires the replacement of the web's infrastructure will not be adopted; a system that ignores the needs of the user will not be used; and a system that assumes the information provider will not attempt to deceive the user will be rendered useless. Exacerbating the problem is the scale of the web's growth, and the need to

solve the information management dichotomy. As such, this represents the problem that this research programme has set out to solve.

The web is an organic system, complex and dynamic, and evolving according to the needs of the user, with the information providers engaged in hyper-competition trying to attract as many users as possible. It is more like a society than a rigid information system, but this is to be expected, as it has virtually no barriers to entry, and so all areas of society contribute to it. As Berners-Lee puts it, "the web is a social creation not a technical one" (Berners-Lee and Fischetti, 1999, p133). As such, this thesis is based on the assumption that in order to manage the web's information effectively, a new and entirely different model of information flow is required, which is sympathetic to the web's existing architecture, the behaviour of its users, and of its information providers. Rather than looking at information from the perspective of the network, the model should focus on information from a human-oriented perspective, as the problems of the web are as much to do with the behaviour of its users as they are with the flaws in its architecture. Such a model has been developed as part of this research programme, and the remainder of this thesis will discuss its design, development and implementation.

4. HOMINID – A Model for Managing Information Flow on the Web

Having discussed at length three core flaws of the web, this chapter presents a new model for managing information flow on the web. The model is called HOMINID, and has been designed to fix the three flaws of the web without falling foul of the information management dichotomy.

4.1 Introduction

This chapter presents a conceptual overview of HOMINID, a new model for managing information flow on the web. The design decisions that led to the development of HOMINID are a result of previous research that has been conducted, but which is not presented in this thesis. Specifically, HOMINID has been designed using the philosophy of a new model of information flow called *Situated Memetic Theory* (SMT), which models information flow from a human perspective. As such, the HOMINID model treats information from the perspective of the user, not the network, and so views information as flowing from an information provider (i.e. the resource owner) to an information consumer (i.e. the user browsing the web). It is for this reason that HOMINID derives its full name: the Human-Oriented model for Managing Information Flow on the web. As this chapter will reveal, the model provides a novel perspective on the nature of information management, and is used to solve the three flaws of the web discussed in the previous chapter, as well as to resolve the information management dichotomy. The research performed in the development of SMT is a significant body of work in its own right, which has led to the development of a powerful new model of cultural information flow. However, the work will not be discussed in this thesis, but will instead be published in appropriate journals.

The chapter presents a conceptual overview of HOMINID, and shows how its core components are able to fix the identified flaws of the web. The remaining chapters discuss how the model has been applied to the web through the development of a new extension to its architecture that fully implements the components of the model. Chapter 6 presents a

prototype of the design, which has been developed to validate the design, and to provide performance data to illustrate the practicality of the model.

4.2 The Core Components of the HOMINID Model

The HOMINID model has been designed to solve the problems of link rot, the shrinking namespace, and increasing noise in the web. In order to achieve this, the model comprises three components, which together fix these three flaws in the web's architecture, and resolve the information management dichotomy. Specifically, the core components of HOMINID include:

- a new scheme for referencing resources across time and space;
- a new resource migration mechanism that migrates resources across time and space;
- a new system for reducing the noise in the web by providing universal access to the web's navigational and characteristic information.

The focus of HOMINID is on the web's hyperlinks; on extending their functionality, redefining their semantics, and ensuring their referential and informational integrity. The following sections describe how this is achieved.

4.3 Reducing Link Rot

Section 3.2.4.1 described the various resource migration mechanisms that have been designed to prevent link rot, and went on to discuss the semantic ambiguity that exists within these systems. This manifests itself whenever a resource's content changes, as it is unclear whether or not the new resource should be given a new URL of its own, or whether it should keep the

URL of the resource that contains the content that it replaces. Essentially, the decision rests on what exactly it is that the URL references: the resource or the content within it. If the URL references the resource, then it will persist throughout the resource's life, regardless of how often the content within it changes. If, however, the URL references the content, then a new URL must be given each time the content changes significantly¹.

From the perspective of the HOMINID model, the URL should be seen as referencing the content of a resource. Recall that the perspective of the model is at the level of the human user, with information viewed as flowing from information provider to information consumer. As such, the user will differentiate between two completely different versions of content, and will see two separate entities, each with distinct identities. The example given in section 3.2.4.1 was the content contained within the front-page of a daily news web site. So, for example, content describing a foot-and-mouth outbreak one day will be distinguished as being completely different from the next day's content, which could, for example, describe the collapse of the deal to build the new National Football Stadium. As such, a human views the two separate pieces of content as completely distinct entities. The web, in contrast, treats them both as one entity: a resource, whose content just happens to change day by day.

¹ Note that what constitutes a significant change of content, worthy of a new identity distinct from the original, is a deep philosophical issue in its own right. For example, should a minor typographical correction be seen as new content? As such, the present work will not attempt to define what is and what is not a significant change of content, and will instead assume that the content author is capable of making up her own mind and assign new identities to the various versions her work has she sees fit.

What this shows is that both link rot and content change effectively destroy the value of a hyperlink from the perspective of the user. Link rot may break a hyperlink from a technical perspective, but content change breaks it from an informational perspective. For example, if another resource references the article about foot-and-mouth by including a hyperlink to the news site's main page, the hyperlink's informational value will be rendered useless the next day when it leads the user to a story about a football stadium.

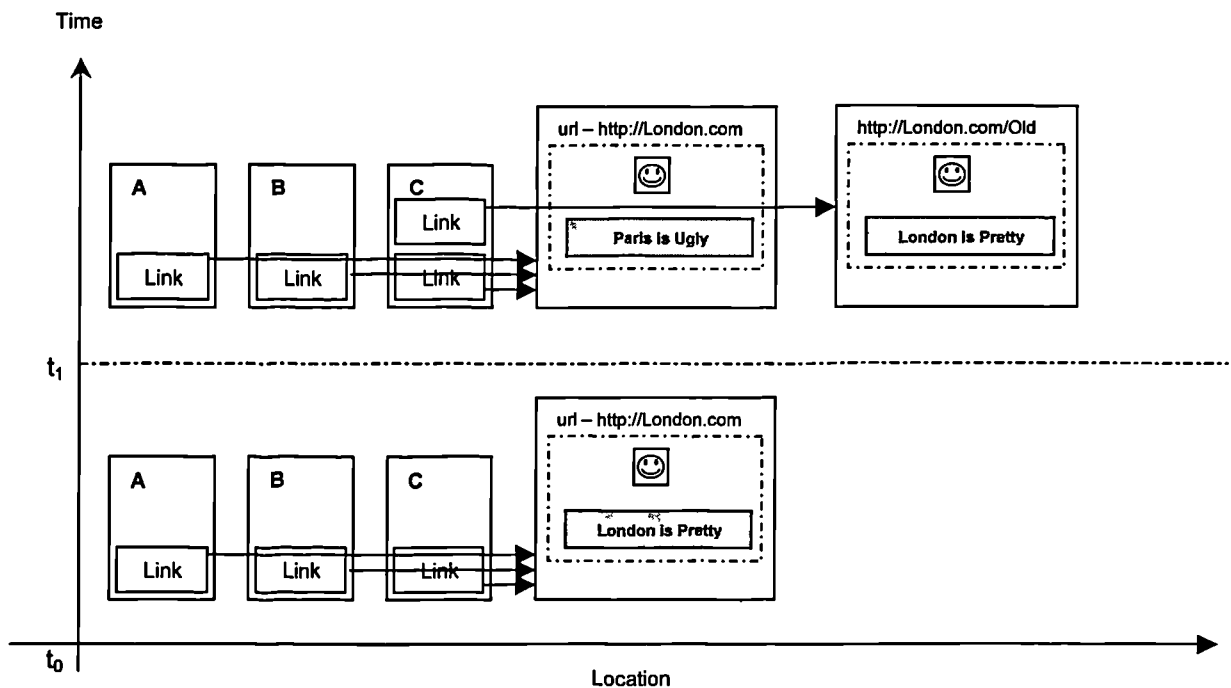


Figure 6 - The Result of Content Changing Within a Resource

From the perspective of the HOMINID model, the problem exists in both cases because the *content* contained within a resource migrates without the knowledge of the set of hyperlinks that references it. This is illustrated in Figure 6, which shows the same resource at the location *http://London.com*, with three hyperlinks referencing it, at two separate points in time, t_0 and t_1 . At t_0 , the hyperlinks in the resources labelled A, B, and C, all reference the resource

at the location *http://London.com*, which contains content relating to the city of London. Clearly, if the resource migrates to a new location at time t_1 , then the hyperlinks referencing it break. However, as Figure 6 illustrates, they are also broken from an informational perspective when the content changes at t_1 , as it is no longer about London; rather, it is now about Paris. In this example, the old London content is archived in a new resource at t_1 , with a different URL (*http://London.com/old*). Effectively, the content contained within the resource at *http://London.com/* has migrated to the resource at *http://London.com/old*, while the existing resource that the set of hyperlinks references remains at its original location. As such, for the integrity of the hyperlinks to be maintained, the HOMINID model must manage resource migration and *content migration* caused by content change.

4.3.1 *Managing Content Migration with Temporal References*

From the perspective of the HOMINID model, the problem of content migration exists because the resource and its content is not treated as an atomic unit; rather, they are treated as separate components, with content forced to migrate away from the resource in which it was originally contained whenever new content in the resource is added. As such, the only way to preserve the integrity of the hyperlinks is for the URL (or other identifier) to reference both the resource and the content encoded within it as a single atomic unit.

Note that this does not contradict the web's current definition of the URL. Although URLs currently reference resources not content, RFC 2396, the current standard for the URI, simply defines a resource as "...anything that has identity." (Berners-Lee et al., 1998, p2). As such, the HOMINID model simply views significantly different versions of content as having

separate identities, and so should be seen as separate resources in their own right. Effectively, RFC 2396 can be seen to agree with both HOMINID and the web, depending upon the perspective from which it is viewed. RFC 2396 claims that anything with identity is a resource. From the perspective of the web, content as we see it does not exist, and so cannot give be given an identity. Effectively, only the set of bits that encode the content can be given an identity and classed as a resource. In contrast, the human-oriented perspective of the HOMINID model *is* fully aware of the content, and so can give it an identity². In this way, both the model of the web and the HOMINID model are consistent with RFC 2396, differing only in the entities that can be given identity. The HOMINID model thereby provides a novel perspective on the architectural standards of the web.

The HOMINID model treats identifiable content and the resource that contains it as a single atomic unit. However, this means that a resource cannot change its content: it must contain it forever. If the content needs to change, it must move with its resource to a new location, and the URL must persist with it. Equally, whenever a resource migrates to a new location, its URL must persist. As section 3.2.4.1 made clear, the problem with this approach is that it requires a resource to use a new URL each time its content changes, which causes a proliferation of new URLs, and places even more pressure on the shrinking namespace. However, the HOMINID model solves this problem by adding the dimension of *time* to the web. The URL is a spatial locator, and so is only able to differentiate between resources at separate physical locations. However, when content changes, the location of the resource

² Note that for clarity, the terms *content* and *resource* will still be used according to their present usage with respect to the web.

remains the same, and *time* becomes the differentiating factor that separates the two resources. Thus, introducing time into the web's references enables two entities to exist at the same point in space, but at different points in time.

To achieve this, a temporal component must be designed for the URL, enabling it to become a *temporal* reference as well as a spatial one. In this way, the resource and its referring hyperlinks are all tightly bound and consistently referenced by the same temporally-enhanced URL. As such, this new temporal referencing scheme is one of the central components of the HOMINID model.

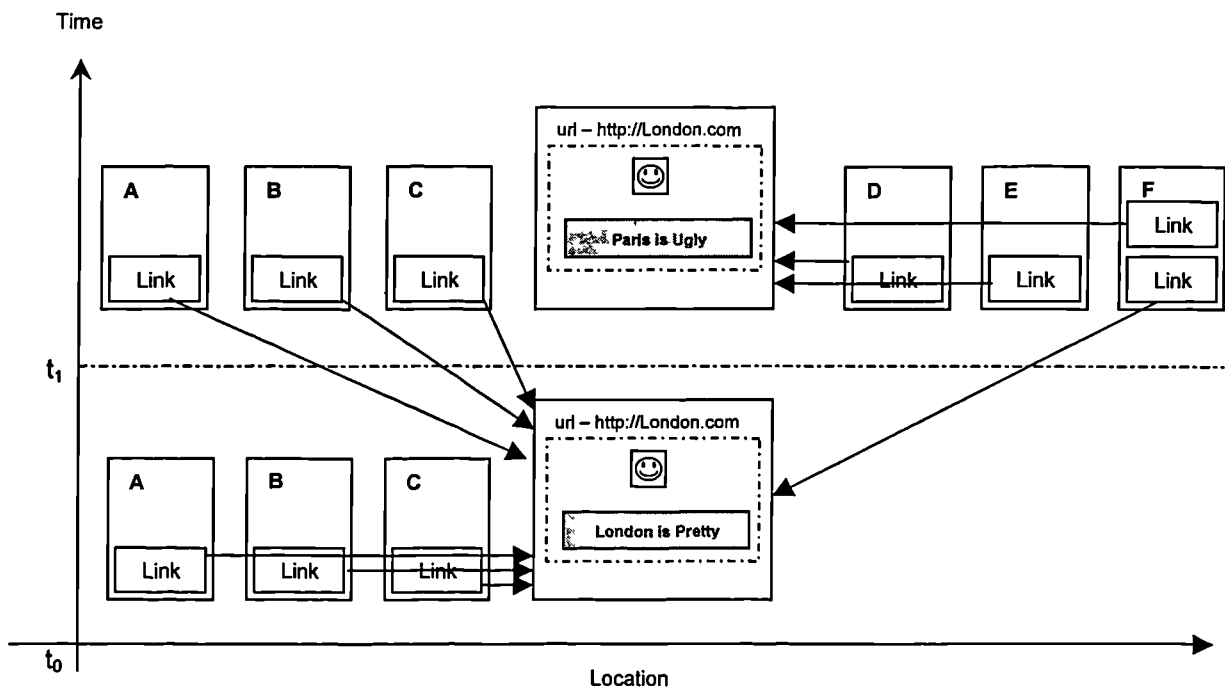


Figure 7 - Temporal Referencing

Figure 7 illustrates the concept of temporal referencing. The figure again shows the location of resources at two discrete points in time, t_0 and t_1 , with the content of the central resource changing at t_1 . However, whereas traditional hyperlinks would reference the wrong content at this stage, temporal references are able to differentiate between the two different resources according to their different locations in time. This enables the original content to remain at its present location (*http://London.com:t0*), with its resource and set of referring hyperlinks intact, while a new resource is created, with new content and new hyperlinks, at the same point in space, but a new point in time (*http://London.com:t1*). Both versions of the content can be referenced by a temporal reference without conflict, and the hyperlinks that reference both resources will remain intact. This is shown in Figure 7 by the new set of referring hyperlinks (i.e. the hyperlinks in the resources D, E, and F) of *http://London.com:t1* co-existing with that of *http://London.com:t0*. Note how Figure 7 shows hyperlinks referencing across the $t_0 - t_1$ boundary, effectively referencing across time. This clearly differs from Figure 6, where only horizontal, spatial referencing was possible. In this way, temporal references not only preserve the referring hyperlinks, they also enable resources to be preserved, providing the web with the means to archive its information, and to enable it to be referenced according to its time of creation. In this way, temporal search engines can be developed that will enable users to search through the web's information archive according to a specific point in time.

4.3.2 Managing Resource Migration with the Resource Locator Service

Temporal references are a fundamentally new approach to resource addressing, but they cannot be supported by the DNS, and so require a new name resolution system. Equally, if the integrity of a hyperlink is to be preserved when the resource it references migrates, the new

name resolution system must provide a mechanism for transparent resource migration. The HOMINID model provides this in the form of the *Resource Locator Service* (RLS), which has been designed as part of this research programme.

The RLS is a system for locating resources on the web across time and space, regardless of how often their locations change. The RLS maps a static resource identifier onto a dynamic location, and ensures that the location is updated whenever the resource migrates or its content changes. The web's hyperlinks can then reference the static identifier rather than the dynamic location, ensuring that the integrity of the hyperlinks throughout the resource's lifetime. In this way, the RLS provides a solution to link rot, and preserves the integrity of hyperlinks throughout resource migration and content change.

The RLS provides a transparent resource migration mechanism that differs from existing mechanisms in two key ways:

- *Resources can be migrated across time as well as space*

The RLS supports temporal references, and so the name of a resource can be mapped to a dynamic position in time as well as space.

- *Resources can be automatically migrated across servers using a remote client*

The RLS provides an interface for remote operation, enabling a resource to be migrated automatically by a remote application.

The RLS represents the most important component of the HOMINID model. However, although it has been designed to replace the web's usage of the DNS, it must still work within the existing architecture and constraints of the web if it is to be successfully deployed. The service is therefore required to:

- provide complete referential integrity for web resources;
- be fully scalable;
- be backwards compatible, such that all web entities (e.g. browsers, servers, etc.) can use the service without change;
- have resolution granularity at the level of the individual resource rather than a host;
- be dynamic such that name or location changes can be made rapidly and automatically;
- implement temporal references by storing details of a resource's name, location, and time of creation;
- provide the location of a resource given a name;
- provide multiple locations if a resource has been replicated.

Designing a service that satisfies these requirements has proven to be a significant engineering challenge. The novel design and implementation of the RLS are presented in the following chapters.

4.4 Easing the Namespace Pressure

As the RLS is designed to replace the DNS, its namespace is free to be defined according to whatever requirements are necessary. As such, this section defines the scope and the

semantics of the RLS's namespace, which has been designed to avoid the problems of the DNS's namespace.

4.4.1 Shrinking Namespace Increases Pressure

The shrinking namespace problem has occurred because the number of *desirable* domain names is vastly smaller than the number of *available* ones. As was discussed in section 3.3.1, there is a premium on memorable names, or those that represent company names.

Domain names must compete against each other for the attention of users, if the web site that they address is to be noticed. Without attention, the information conveyed by the web site will simply be ignored. For e-commerce sites, this is financially devastating. A good domain name, therefore, can be extremely valuable.

Receiving attention through a good domain name is exceptionally difficult, however. Entering a URL into an address bar accounts for only 2% of all navigation events (Catledge and Pitkow, 1995). Compounding this is the sheer number of domain names competing for this limited attention. However, the value of a good domain name can be illustrated by the fact that of those people who bought goods online, some 60% did so by entering the URL directly into the address bar (SRI, 2000). As such, although only a small proportion of navigation events directly involve the address bar, those that do are extremely valuable.

From the perspective of the web site owner, therefore, any strategy for creating a memorable domain name has the potential to dramatically increase the number of visitors to the site. For

example, strategies that encourage the user to pass the domain name onto friends, or which create domain names that are more memorable, or easier to type, will be more successful than those that lead to obscure, meaningless, or syntactically awkward domain names. This is the reason that most of the words in the dictionary have already been registered, and domain names such as *drugs.com* are commanding \$1,000,000 (Arent, 1999).

However, although most of these desirable names are already registered, or are so expensive as to be out of the reach of most people, it has not meant that all other domain names have been rendered useless. Rather, the shrinking namespace has forced new strategies to evolve and the resultant domain names to adapt, leading to domain names that try to gain attention using whatever strategy works. Unfortunately, as the following section shows, some of these strategies benefit the web site owner at the expense of the user.

4.4.1.1 Exploitative Strategies

Few users are actually good at typing, and so regardless of the ease with which a domain name can be typed, mistakes will happen. As such, one common exploitative strategy used in the creation of domain names is to adopt a name that is syntactically close to a very well-known existing domain name, but which differs by one or two letters that match common typing errors made by users. In this way, the domain name acts as a parasite, living off the attention that should belong to the well-known domain name. For example, AltaVista's site is very popular, and large numbers of people regularly type its domain name, *www.altavista.com*, into their address bar. Exploiting the user's typing error, however, is the URL *www.atlavista.com*,

which opens up a separate browser window and redirects people to the sites *www.tickerprofiles.com/profiles/liquidics/* and *www.otcstreet.com/trivia/otctrivia.cfm*.

Another exploitative strategy is to hijack existing trademarks and essentially steal attention from the trademark owner. Well-known brands, such as McDonalds, are so pervasive that a user typing in the domain name *www.mcdonalds.com* would automatically expect to see that company's web site. In this way, the well-known domain name will be almost guaranteed a large amount of attention. As such, if a company other than McDonalds registers the well-known domain name before McDonalds themselves do, they can be guaranteed of this attention, regardless of the relevance of their site to the domain name.

4.4.1.2 The Problem With ICANN's Solution

From this perspective, ICANN's solution will only work in the short-term, if at all. Domain names are constrained into a strict namespace, which limits the number of desirable names. Expanding the number of TLDs does not free the namespace, and so less-memorable domain names will have to adopt the same exploitative strategies within each new TLD if they are to receive any attention. However, the situation could be made worse, as the new TLDs act to classify each domain name according to a specific category. As such, domain-specific TLDs, such as *.museum*, will act to *increase* the pressure on those domain names that belong to each category, by implicitly identifying the information need of the user, and thus rewarding the parasitic strategy even more.

For example, suppose a user wishes to navigate to the web site of the Natural History Museum, but accidentally types *www.nmh.ac.uk*, rather than *www.nhm.ac.uk*. With the current TLDs, it is difficult to determine what the user navigating to a specific site is actually looking for, and *NMH* could represent virtually anything. As such, the content of the site behind a parasitic domain name must be general, such as a gambling site, if it is to attract much attention, as focusing on a specific subject that it assumes the user is looking for could deprive it of a large amount of attention if its assumption is wrong. As such, it is better to play it safe. In this way, the user's mistake will inadvertently take them to a web site that is clearly different from the one they intended to visit, and their mistake will be obvious.

However, with a highly focused TLD such as *.museum*, it is obvious what the user is looking for: a museum. Now, it is perfectly safe for the content of a web site behind a parasitic domain name to focus on a specific subject. In this example, a parasitic web site could use the new domain name *www.nmh.museum*, and sell items that are also sold at the Natural History Museum, but which are much cheaper at the parasitic web site. As such, the user has been inadvertently taken to a direct competitor to the museum, and even if the mistake is realized, the user may decide to stay anyway, as the content behind the parasitic domain name will be relevant to their needs. In this way, the strategy of the parasitic domain name will become even more successful, and encourage imitators to adopt the same strategy for their own domain names. Effectively, the new TLDs will encourage *more* domain names to become parasitic, not less, particularly once the desirable names in each TLD have been exhausted. As such, ICANN's solution will only exacerbate the problems it set out to solve.

4.4.2 *Easing the Pressure*

If the HOMINID model is to resolve the namespace problems that affect the DNS, it must ease the pressure on the namespace, taking into account the adaptive pressure that will be placed upon it. As such, it must be free enough to discourage exploitative strategies that deceptive domain names will adopt. However, it is doubtful that exploitative strategies can ever be fully prevented, but they can be discouraged by making them less rewarding. This can be achieved by opening up the namespace so that it uses any name; that is, the RLS should have a *completely unconstrained* namespace, mapping any string representing a name onto any string representing a location.

In this way, the pressure on the namespace can be relieved by dramatically increasing the number of potential names that can be used. Parasitic strategies will still be employed, as the attention generated by popular names and brands will always attract such parasites. However, by opening up the namespace, there will not be such pressure overall, and there will be more than enough space for new desirable names to exist, which otherwise would not under the DNS's strict syntax. As such, other, more positive strategies will become more successful, and the parasitic strategy will become less successful. In this way, the problems of exploitative and deceptive strategies are not eradicated, but they are made less successful, which should drastically reduce the incidence of such names over time.

4.4.3 *Defining the Semantics of the New Namespace*

The introduction of a new name resolution service with its own namespace will take the pressure off the DNS, such that its semantics can revert to their original specification (i.e. IP

address/hostname resolution). The unconstrained nature of the new namespace should enable the definition of new *sub*-namespaces that exist within it. The relationship between these sub-namespaces and the global namespace is similar to that of the URL and the URI³, but with the exception that the new namespace will impose no restrictions on the sub-namespaces, other than the requirement that each name must be unique within the global namespace.

In this way, the semantics of the new namespace are perhaps closer to that of the CORBA Naming Service. Recall from Table 2 that CORBA does not manage or even attempt to understand the syntax used in its Naming Service's *kind* attribute, but leaves it to higher levels of software, which can impose their own management policies on the naming of objects (OMG, 2000). In a similar way, the RLS is only required to map the name of a resource onto its location within the web, and should leave the semantics behind the namespace to whichever naming policy is in use.

In this way, the RLS becomes a very flexible name resolution service, which can be shaped according to the requirements of its users. However, because the namespace is not constrained, the name are free to evolve outside of any sub-namespace, if required, keeping the namespace pressure low without restricting the usefulness of the new system. For example, the namespace of the DNS can be seen as a sub-namespace of the RLS, but if it runs out of names, a new sub-namespace can be defined and implemented without any change to

³ Recall from section 2.2.2.2 that the syntax of the URI defines the structure for all web identifiers, and so represents the superset of web-identifiers.

the RLS. This essentially makes the new service future-proof, bound only by the technology upon which it runs.

Note that hierarchical naming schemes are used throughout distributed systems of all types, largely because they manage very large namespaces very efficiently. As such, a flat namespace, which the RLS's namespace represents, may prove unmanageable in the long term. However, although the RLS supports any string as a valid name, it can easily be adapted to restrict the set of names it supports to one or more namespaces, the introduction and management of which can be controlled by an organization such as ICANN. Effectively, the RLS has been designed to remove all *technical* limitations from the design of namespaces on the web; how the namespace is used then becomes a matter of policy. Future research will focus on the effect an unrestricted namespace has on the naming conventions of web resources, to determine whether some restrictions are necessary. However, whatever the conclusion of this research, the RLS is flexible enough to provide a platform that supports *any* naming policy, and it is this flexibility that is one of the key innovations in its design.

4.5 *Reducing the Noise in the Web*

In the same way that domain names must compete for the attention of a user, so too must web resource if the web sites that they are part of are to become successful. The predominant business models for web sites are currently based on advertising or e-commerce, both of which define success according to the number of people who visit a site; without visitors, advertising will not work and products will remain unsold.

Analysing the web from this perspective, noise can be seen as almost inevitable. Users may wish to seek out specific web resources, but each resource will do all it can to receive the attention of *any* user, regardless of their information need. A resource thrives on attention, and so a popular resource leads to a healthy site, particularly when its business model is based on advertising.

However, advertising does not have to be relevant to the information need of the user browsing the web. As such, it pays a web resource owner to attract a user to his resource, irrespective of its relevance to the user's information need. In this way, relevance can be seen almost as a hurdle to the business prospects of the web site, which may actively employ deceptive strategies to circumvent any relevance heuristics used by the user and the search engine. The following sub-sections provide an in-depth analysis of this deception, and show how it increases the noise in the web by impacting the integrity of its navigational mechanisms.

4.5.1 The Deceptive Hyperlink Versus the User

A web resource's chances of attracting attention will be greater if the hyperlinks that reference it can attract more attention than those of its peers. As is the case with the domain name, however, the pressure is on the hyperlink to attract attention using *any* strategy that works, including those that are at the expense of the user. Unfortunately for the user, unless the hyperlink is from a trusted and well-known source, there is no easy way for them to discern the strategy that the hyperlink employs, other than actually clicking on it. Worse, the user cannot go back to the hyperlink and rate it as trustworthy or not, in order to prevent other users

from making the same mistake, and so the hyperlink is free to carry out whatever strategy it likes with impunity.

4.5.1.1 Deception as an Effective Strategy

There are many different types of deceptive strategies employed by hyperlinks, and most predate the web. For example, hyperlinks usually contain text that is supposed to describe the content of the referenced resource, but there is no mechanism to guarantee this constraint. As such, it is easy for the hyperlink to deceive the user by describing the content in false terms. A more elaborate example of this is the hyperlink that hides behind an image of a user interface control, such as a button. Such a hyperlink, called a Fake User Interface, or FUI, is designed to deceive the user into clicking it by pretending to actively control elements of the web page in which it is hosted. Of course, no such control is provided, and clicking on it presents the unwary user with an unwanted resource.

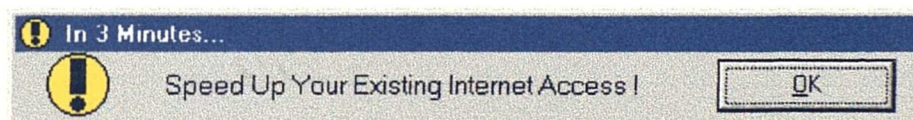


Figure 8 - Real Example of Fake User Interface

Figure 8 provides a real example of a FUI from the company Bonzi (www.bonzi.com). The figure shows a banner advert that seems to show a button, but the whole image simply directs the user to the Bonzi web site. This FUI is so effective that it was NetRatings' most-clicked banner advert on the web in February 1999 (Cox, 1999). It has also spawned many imitators (Figure 9).

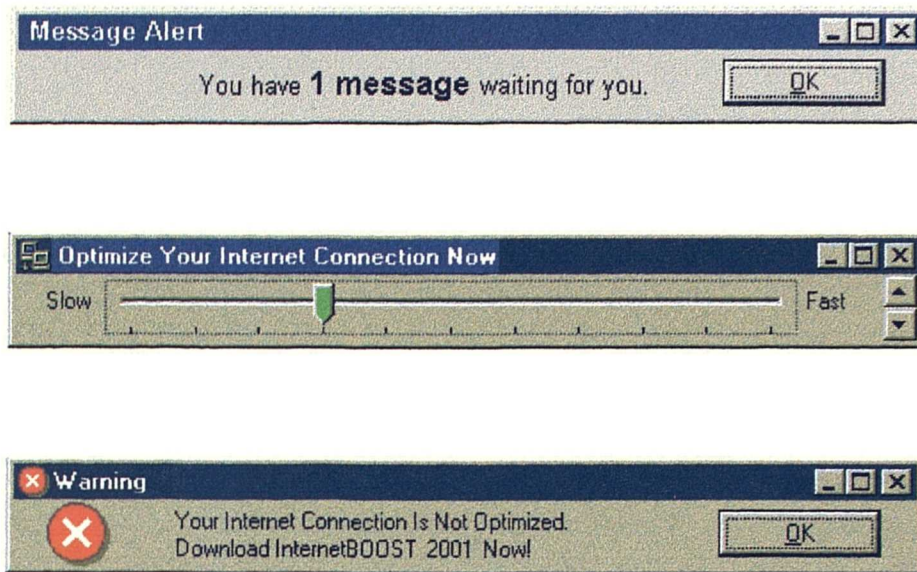


Figure 9 - Fake User Interface Imitators

A further example of the deceptive strategy can be found with the various navigational cues provided by a web browser. For example, when a user places the mouse cursor over a hyperlink, the URL of the resource that the hyperlink points to appears in the browser's status bar (see Figure 10). If the URL contains the same domain name as the current web page, the user can be reasonably sure that clicking the link will take her to another web page in the same site. However, deceptive hyperlinks insert their own text into the status bar, which either hides the URL of the actual resource that the hyperlink points to, or pretends to be an honest-looking URL when in fact it redirects the user to a different web site.

In all of these examples (and this is by no means an exhaustive list), the user is deceived by the hyperlink into giving attention to an unwanted web resource, but has no means of reproaching the hyperlink. Thus, the web resource that is referenced by the hyperlink gets

more attention than it would ordinarily, making the deceptive strategy employed by the hyperlink is an unqualified success.

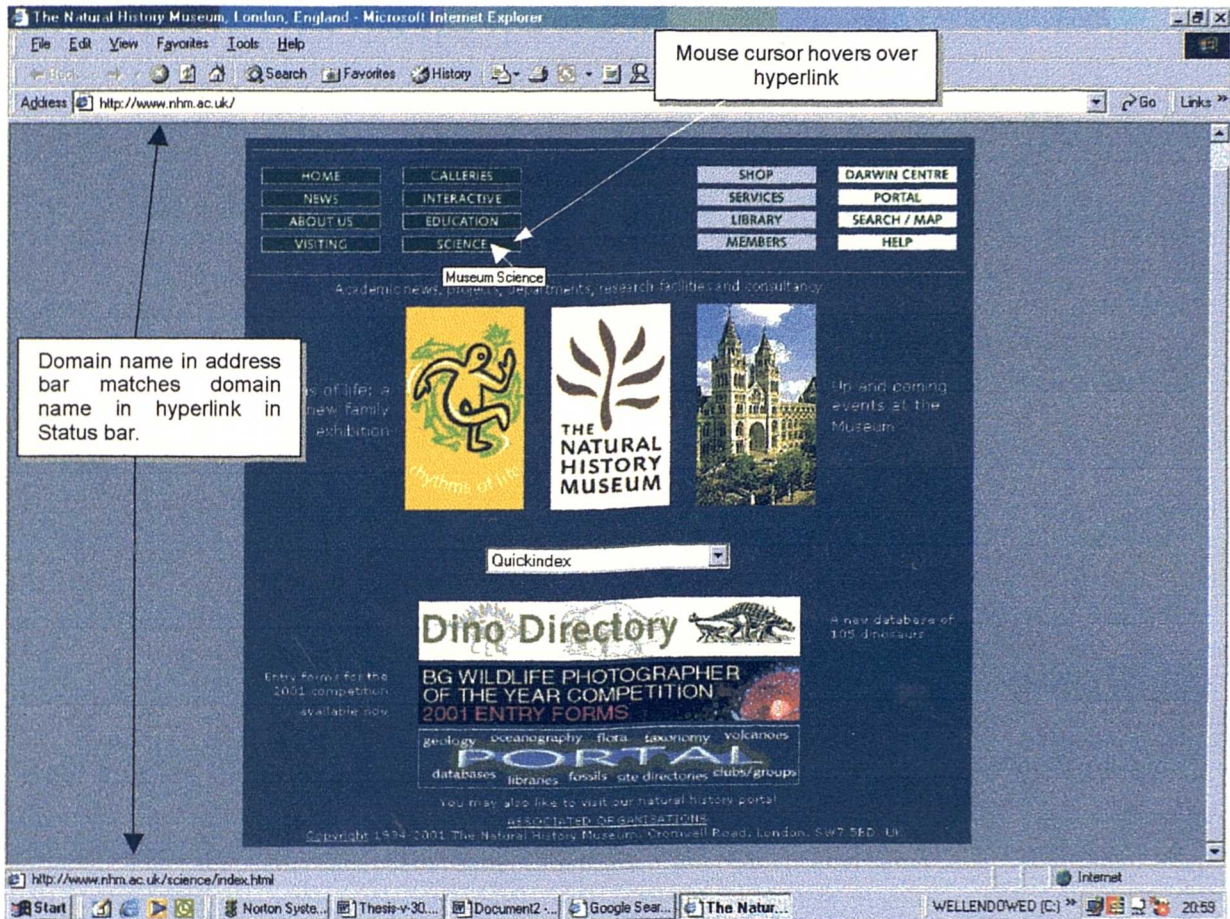


Figure 10 - The Browser's Status Bar as a Navigation Aid

4.5.1.2 How the Hyperlink Breaks the Flow of Information

There are many reasons why a user clicks on a hyperlink. Obviously all users' navigational heuristics are different, but there are common types of heuristic that can be identified. For example, a user may click on a hyperlink if it is of a type that she perceives exhibits authority

(Kleinberg, 1998), or if it appears to satisfy her information need. Fundamentally, the reason for choosing one hyperlink over another is based on implicit information that the user has about the hyperlink, the resource it points to, the environment in which the hyperlink is situated, and information the user already has about the world. The constraints that the user is attuned to between these different situation types cause information to flow, and it is this information that helps her to select one hyperlink over another. However, it is these constraints that also represent the user's navigational heuristics, and so it is in a web resource's best interests to exploit them.

The success of a user's navigational heuristics is dependent upon the user being aware of the hyperlink's type, and attuned to the constraints that all hyperlinks of a specific type are worth clicking. For example, a user must be aware that a hyperlink is of an authoritative type, and attuned to the constraint that authoritative hyperlinks are worth clicking. However, the implicit constraints that enable the user to recognize these types can be manipulated freely by the hyperlinks themselves. For example, a hyperlink *should* display the URL of the resource that it references in the browser's status bar, but it does not have to; equally, it *should* provide textual or visual information that represents the resource, but it does not have to. Consequently, the hyperlink can present whatever information it wishes, and can therefore present information specific to a *reliable* browsing situation, when in fact the user is unwittingly placed in an *unreliable* browsing situation. In this way, the constraints that the user will be attuned to, and which she will use when judging the hyperlink, can be hijacked by the hyperlink, and will lead to *misinformation*.

Ultimately, it is easy for a poor quality hyperlink to attract attention through deceiving the user, and because it attracts more attention without incurring any penalties from the user, it is a good strategy for a resource to employ. Deceptive hyperlinks that reference low value resources will, on balance, attract more attention over time than reliable hyperlinks, because it takes less effort to create a dishonest hyperlink than it does to create a resource of high informational value. As such, the selection pressures on the hyperlinks will cause their strategies to evolve to become more dishonest, until the constraints that the user uses to select them break down completely.

The only defence the user has is to learn from the deception, and to recognize a deceptive strategy when one presents itself. However, the deceptive strategies are not static, and will constantly evolve to continue to deceive the user. Effectively, an arms race is set in motion between the user, who must determine the type of the hyperlink situation, and the hyperlink, which must fool the user by exploiting her constraints. As the hyperlink is the primary mechanism for navigation on the web, however, this is a serious state of affairs for it to be in. Effectively, the key goal of the web's primary navigation mechanism is to *deceive* the user into going somewhere they do not want to be.

4.5.2 The Deceptive Web Site Versus the Search Engine

Users are not alone in their battle against deception. Search engines, too, are prey to this strategy, as they too provide a web site indirectly with attention. Because search engines are so popular (see Figure 11), they can direct vast amounts of attention to a site, but only if the site appears high up in a search engine's results list. As such, it is in a site's best interests to

encourage the search engine to perceive it to be as relevant as possible to the user's query, even if that means deceiving the search engine. However, because a search engine must use static, hard-coded heuristics when determining relevance, it cannot respond to the strategies as quickly as they evolve, and so resources within a site can easily pretend to be something they are not, or more relevant to a query than they actually are. In this way, the quality of the search engine is seriously weakened by the deception of the site.

Search Engine	Searches per Day
AltaVista	50 million
Inktomi	47 million
Google	40 million
GoTo	5 million
Ask Jeeves	4 million
Voila	1.5 million

Figure 11 - Number of Queries per Day for the Popular Search Engines (Sullivan, 2000c)

4.5.2.1 An Arms Race Between the Search Engine and the Web Resource

From the user's perspective, a search engine should take a query, and return only those web pages that are completely relevant, and rank them in order of relevance. However, from the site's perspective, the search engine should return a reference to only its resources, and no others. As such, it is a good strategy for a resource to deceive the search engine's relevance heuristics into ranking it as highly relevant for as many queries as possible. As search engines become wiser to a resource's deceptive strategy, so the resource must adapt its strategy if it is

to survive. The result is an arms race between the resource and the search engine, with the relevance heuristics of the search engine representing the battlefield. At stake are the quality of the search engine and the integrity of one of the web's primary navigation mechanisms.

4.5.2.2 The Fight For Relevance

Originally, most search engines determined relevance simply by counting the number of times the words in a user's query appeared in a HTML document, and used that to classify the document's relevance to the query. However, the strategies employed by web site owners soon evolved to adapt to this, and they began to embed the same popular keywords many times into a document, regardless of its relevance to the query or the keyword. This 'search engine spamming' as it is known (Lawrence and Giles, 1998) is designed purely to improve the document's ranking in the search engine, causing it to be placed at the top of the search engine's results list across a wide range of queries, and so giving the document more attention.

These fake keywords were originally displayed at the bottom of a web page in a simple list, with a large amount of space between them and the actual content of the document. In this way, the user would never see them, but the search engine would index them. However, search engines retaliated by ignoring words below a certain point in a document. The web site owners responded by placing the fake words at the start of the document, but in a font so small that the user could not see it (so called *tiny text* (Sullivan, 2000d)). When the search engines adapted to this, the owners responded again by making the fake keywords the same colour as the background, rendering it invisible to the user.

In an attempt to bypass the spamming keywords problem, newer techniques focus on the user. The experimental Inquirus meta-search engine (Lawrence and Giles, 1998), for example, queries a number of search engines on behalf of the user (hence the term *meta*-search engine), and returns not only the title of each document and its URL, but the text that surrounds the keywords. Inquirus relies on the user determining the relevance of each document for themselves, but without having to download the document first. However, this could be seen as an abdication of responsibility, as the whole point of a search engine is to perform the search on behalf of the user, and with the number of documents returned by most search engines, the user must be prepared to put in a lot of effort.

Other search engines that focus on the user provide filtering technology, such as Northern Light (www.northernlight.com). Filtering gives the user the chance to edit their query by selecting criteria that each document should be judged against (for example, documents from a specific location) (Chakrabarti et al., 1999b). In this way, the user determines the relevance heuristics for themselves. The advantage of this approach is that the relevance heuristics are dynamic, thus thwarting any strategies designed to exploit the more traditional, hard-coded heuristics. However, the disadvantage is that it relies on the user being relatively expert with information retrieval techniques, with the result that the novice user could filter out useful results (Chakrabarti et al., 1999b).

The latest attempt to defeat the deceptive strategies is to rely on the link structure of the web to provide an indication of the *authority* of a document. This technique ranks a document according to both its perceived relevance, judged according to normal heuristics, and its

authority within the web community, judged by the number of hyperlinks pointing to it. Google's PageRank uses this method (see section 2.3.3.1), as does IBM's Clever, which draws on the work of Kleinberg (1998), who, like Google's Brin and Page (1998), views the hyperlink as conferring authority on the page it links to. However, unlike Google, Kleinberg also defines the notion of Hubs, which are pages that link to many authorities. His HITS algorithm (Chakrabarti et al., 1999b), which underlies the Clever search engine, differs from Google's PageRank algorithm in that it is able to identify good hub pages as well as authoritative pages, but the principles underlying each algorithm are similar. In this way, it does not matter what strategy the HTML document tries to use to fool the search engine, as the document's ranking in the search engine is determined purely by the number of links that point to it, and these links are dependent upon the co-operation of many other users. As such, Google and Clever assume that it is more difficult to co-ordinate deception across web sites than it is from one web site.

However, it is not impossible. Web sites can encourage many hyperlinks to point to them through advertising (Kleinberg, 1998), thus literally paying for attention. Alternatively, with the cost of web publishing so low, artificial hyperlinks can be created in the form of many separate web sites, which exist simply to provide a spurious link to the resource that may never be used, but which will be counted by the search engine (see section 3.4.3.3). This artificially inflates the resource's presence on the web, and thus deceives the search engine.

Google and Clever are the latest in a long line of search engines, each of which has tried to outsmart the deceptive strategies employed by web site owners, but which has been

outmanoeuvred by strategies that can adapt faster than an engine's heuristics. It is expensive to develop a search engine from scratch, and difficult to modify the heuristics of a mature index. As such, the well-known heuristics of an engine are easy prey for the fast moving strategies of the web site owner.

4.5.3 A Persistent Problem

The deceptive strategies employed by web site owners have clearly had an impact on the quality of the web's information. However, it will also impact its quality in the future, as the same competitive forces will exist to deceive any new technology that is developed to increase the quality. For example, as discussed in section 2.3.2, meta-data formats such as XML and RDF have long been cited as the means to solve the web's information retrieval problems (Lassila, 1997; Heery, 1996). However, meta-data may work perfectly in controlled environments, such as academic journals, but it is difficult to see how any kind of meta-data will lead to higher quality information retrieval when it will be under the direct influence of the web site owner. As long as there is no gatekeeper in the web, relevance will be seen as an obstacle, and deceptive strategies will easily be able to deceive any heuristics. The challenge for the HOMINID model, therefore, is to overcome this problem without introducing a gatekeeper into the web; in short, the HOMINID model must resolve the information management dichotomy.

4.5.4 The Oracle Server – A Novel Platform for Enhanced Navigation

Web sites are able to hijack a user's constraints because they are free to present misinformation to the user via the deceptive hyperlink that is consistent with a relevant situation. To prevent this, the user must be made aware of the situation that they are actually

in, rather than the situation that the resources in a web site try to pretend they are in. The HOMINID model achieves this through applying a component of situation theory called the *Oracle situation* (Devlin, 1991) onto the browsing environment experienced by the user.

An Oracle situation supports all of the information about a situation from the moment of its creation, to the moment of its destruction. Thus, the Oracle situation of a web resource is a completely objective, factual situation that supports *all* of the information about the resource, and not just the information that the resource owner wishes to present. This includes all the characteristic information about a resource (such as its informational content, its creator, the time of its creation, etc.) as well as all the people who have seen it, when they saw it, etc. As such, by being made more aware of this situation, the user does not have to rely on the information presented solely by the resource owner, and so can choose whether or not to provide a resource with any attention based on *reliable* independent information. Effectively, the user consults the Oracle before deciding whether or not to click on a hyperlink.

In this way, the user is made aware of the real situation they are presented with, and so can determine the constraints that are appropriate to this situation, rather than the situation that the resource tries to present. This acts to sharpen the user's navigational heuristics, and prevents her constraints from being manipulated, enabling the deceptive strategies of certain web resources to be seen *prior* to the user paying them any attention. This gives the user the choice of whether or not to give her attention to a resource, rather than the existing approach in which she is forced to give her attention *before* determining its relevance to her information need. As such, the selection pressures imposed on the resource will be those that benefit the

user, rather than those that benefit the site, and so the advantages of the deceptive strategy will be reduced in favour of a strategy that provides more relevant information.

4.5.4.1 Resolving the Information Management Dichotomy

This approach resolves the information management dichotomy by enabling the user to navigate across a set of information more effectively according to her own selection criteria, rather than the censorship approach, which restricts the set of information that exists according to a third party's selection criteria. As such, the two approaches can be seen as opposite methods to achieve the same result: the Oracle approach acts to limit the set of information through which the user must navigate by enabling her to reject information that is not relevant without removing it; whereas the censorship approach acts to limit the set of information by explicitly removing it from the system before she can determine its relevance. The Oracle approach is therefore better, because the information is selected according to the needs of the user and not a third party censor, and the total pool of information that exists on the web is left intact. In this way, control is imposed on the information without requiring it to be permanently censored.

4.5.4.2 Functional Operation of the Oracle Server

The approach of the HOMINID model is to store small amounts of information from the Oracle situation that are pertinent to the selection of the resource, and present it to the user via the web's hyperlinks such that she becomes more aware of the Oracle situation than she does about the resource's (fake) situation. The HOMINID model achieves this through an entity called the Oracle Server, which serves information about the Oracle situation of a resource via the hyperlinks that reference it. As such, the HOMINID model's approach to reducing the

noise in the web is to focus on the *informational* integrity of the hyperlinks, and to enhance the user's browsing skills, rather than to provide an enhanced search engine.

The Oracle Server operates by identifying the hyperlinks that employ a potentially deceptive strategy, and alerting the user before she actually clicks on one. It uses heuristics to infer each hyperlink's strategy from selected, unbiased information about the hyperlink, the resource, and the web site that the resource is part of. The Oracle Server obtains some of this information from the resource owner, who provides *characteristic infons*⁴ about his resources, and the rest from the navigational patterns that emerge from users' browsing sessions, which provide unbiased *navigational infons* that cannot be manipulated by the resource or its owner. If the Oracle Server detects a deceptive strategy, it can alert the user by informing the user's browser, which can display the hyperlink using a different colour, for example, or by greying it out. In this way, the user can see the situation that the resource presents, and also the real situation obtained from the Oracle situation.

4.5.4.2.1 *Characteristic Infons*

The Oracle Server stores characteristic infons about a resource. Characteristic infons are those that describe the characteristics of the resource, such as its subject, its informational content, colour, date of creation, owner, author, file size, etc. Such infons help a user to select a resource based on its characteristic attributes before they navigate to it. These infons are

⁴ Infons are the fundamental unit of information in situation theory, and represent facts about the world (Devlin, 1991).

meta-data that enhance the information that flows from a hyperlink, allowing the user to reject, for example, a resource from an author whose previous resources have employed deceptive strategies. The user can enter into their browser a range of characteristic infons that they wish to be alerted to whenever they navigate to a web page that contains a hyperlink that references a resource with matching characteristics. In this way, the Oracle Server filters out hyperlinks according to the user's own navigational heuristics, and can use these as part of its own heuristics in order to automatically identify deceptive strategies.

4.5.4.2.2 *Navigational Infons*

Although meta-data is undoubtedly useful, it can still be manipulated by the resource, as it is the resource owner who provides it. Thus, an Oracle Server that simply stored meta-data would serve the user no better than current search engines. To resolve this, the Oracle Server also stores *navigational* infons, which are derived from the patterns of users' browsing behaviour. These patterns reveal the way in which a user navigates across the web, and by applying heuristics to the infons contained within them, the Oracle Server can automatically identify a deceptive hyperlink.

Many studies have found patterns in web users' browsing behaviour. Hochheiser and Schneiderman (1999), for example, discovered the emergence of such patterns using interactive starfield visualizations. Huberman et al. (1998) showed that such navigation patterns display "...strong statistical regularities that can be described by a universal law". Pitkow and Recker (1994) found the existence of "...long sequences of between-site access patterns on a per session and a per user basis" (Pitkow and Recker, 1994), and found that

“...in one session, a user visited seven different sites in consecutive order five times.” (Pitkow and Recker, 1994). This could be explained by a study by Tauscher and Greenberg (1997), who found that the probability of each resource having already been seen by a user is 58%, indicating that users have a set of familiar web sites that are appropriate to their information needs, and which they keep returning to. The same study was also able to identify seven distinct browsing patterns from the navigation behaviour of 23 users over the course of six weeks (Tauscher and Greenberg, 1997).

However, the navigational infons are currently contained within the access logs of web servers, which record how many times each resource is downloaded, but extracting reliable information from these log files is notoriously difficult (Pitkow, 1997). For example, each time a user downloads a resource from its host server (termed the *origin server*), the server records the event (termed a *hit*) in its access log. Currently, however, there is no universal access to these logs, as each is kept on the server that maintains it, and so the navigational infons exist, but not in a form that enables cross-server querying. Worse, caches, whether on the client browser, or in a caching proxy server, serve a web resource without the origin server registering any hits, while proxy servers mask the number of users accessing a server, making paths from individual browsing sessions extremely difficult to identify (Pitkow, 1997). The Oracle Server must therefore provide *universal access* to reliable navigational infons from all web servers across the web⁵.

⁵ Note that the Oracle Server should not store any information that can identify a specific user; only the *anonymous* navigation pattern that the user makes.

4.5.4.2.3 *The Heuristics of the Oracle Server*

The heuristics used by the Oracle Server to identify a deceptive strategy are based on the user's browsing behaviour when they have been deceived. Specifically, the user will click on a hyperlink and download the resource that contains the unwanted resource. Once the user sees that the resource is unwanted, they quickly navigate out of the site that hosts it. As such, many users will click on the hyperlink connecting the user's current resource to the resource containing the unwanted resource, as its deceptive strategy is designed to capture attention at all costs (recall from section 4.5.1.1 how the Bonzi deceptive hyperlink, which employed the FUI strategy, became the most clicked banner advert on the web). Such hyperlinks are usually banner adverts, which link separate web sites, and so are called *inter-site* hyperlinks.

However, because the user leaves the site at this stage, no other hyperlink within the site is clicked. As such, the *intra-site* hyperlinks appear to provide enormous resistance to the user, and register very few hits. The deceptive hyperlink can therefore be exposed through a combination of high inter-site hyperlink usage and low intra-site hyperlink usage. In this way, the navigational infons from across web servers act to provide an unbiased view of the strategies of the web's hyperlinks.

Note that the Oracle Server is an open platform, such that its heuristics can be changed as the strategies of the resource owner evolve. As such, the intention is for the simple heuristic defined here to be replaced with more sophisticated heuristics after further research has been conducted into the navigation patterns of users. The Oracle Server should therefore be seen as

a platform for increasingly sophisticated web browsing, which should reduce the noise on the web significantly.

4.5.4.3 New Web Metrics

As well as using heuristics to determine the strategies of the web's hyperlinks, the Oracle Server can also measure the state of the web and its content, and so provide the user with further information on a resource's effective quality. Specifically, the Oracle Server can be used to:

- *Measure the Resource*

The Oracle Server can provide accurate information on how much attention the resource at different sites across the web, allowing the real value of the resource to be accurately determined.

- *Measure the attention flowing through a web site's hyperlinks.*

The resistance of the set of hyperlinks that reference a resource directly affects the amount of attention that the resource receives. The Oracle Server can therefore be used to determine the effectiveness of a resource's hyperlinks, and the web sites in which they are located. This provides the hyperlink and the web site with a real, tangible sense of value. If the hyperlink is clicked frequently, but only in certain web sites, then those web sites clearly provide a suitable environment in which the phenotypic effects of the hyperlink work best. As such, the web sites will become more valuable for that type of hyperlink. In this way, advertising hyperlinks can be

better placed, which should result in advertisements being more relevant to the environment in which they are situated.

- *Measure the Potential Attention*

Universal access to navigational infons enables a resource owner to see the amount of attention that the web sites hosting his resource's referring hyperlinks attract, and compare it to the amount of people who actually click on it. In this way, the resource owner can determine the amount of potential attention that his resource could receive, and adjust the design of the hyperlinks to maximize the attention that is actually received. Equally, if the amount of potential attention is too low, the resource owner can search other sites for other sources of higher potential attention in which to host his resource's referring hyperlinks. In this way, the resource owner acts to situate the hyperlinks in an environment that maximizes the attention that the resource can receive, thus benefiting the resource without deceiving the user.

In this way, universal access to the navigational infons enables the Oracle Server to characterize and measure the resource on the web, thereby providing the user with advance information about the real situation they are in, rather than the situation that the resource tries to pretend they are in.

4.6 *Summary*

This chapter has described the basic components and philosophy of the HOMINID model, which defines a new way of managing information flow on the web. The chapter has shown

how the model fixes the three identified flaws of the web's existing architecture without falling foul of the information management dichotomy. The key concepts of the HOMINID model are presented in Table 5.

Concept	Problem Solved	Description
Temporal Reference	<ul style="list-style-type: none"> • Destruction of the hyperlink due to content migration • Lost History 	<p>The Temporal Reference binds content and a resource together as one atomic unit, and locates that unit in time and space. Should any component of this unit change, it becomes a new unit, and must receive a new temporal reference.</p>
Resource Locator Service	<ul style="list-style-type: none"> • Link Rot • Shrinking Namespace • Automatic, transparent resource migration 	<p>The RLS is functionally equivalent to the DNS, but does not constrain the namespace. Its default namespace is the temporal reference, which enables it to locate a resource across time and space. The RLS also provides a transparent resource migration mechanism that can enable a resource to be migrated remotely.</p>
Oracle Server	<ul style="list-style-type: none"> • Increasing Noise • Ineffective Browsing caused by deceptive hyperlinks • Resolves the Information Management Dichotomy • Web Metrics 	<p>The Oracle Server provides universal access to characteristic infons and navigational infons about the resources on the web and the way in which they are used. In this way, it can measure the resource and its referring hyperlinks, and provides the user with information from the Oracle situation rather than from the resource's deceptive situation, thus maintaining the hyperlink's informational integrity. As such, the Oracle Server can alert the user to deceptive strategies and help them to make more informed browsing choices. This reduces the noise in the web without requiring the censorship of its information, and so resolves the information management dichotomy.</p>

Table 5 - The Core Components of the HOMINID Model

From this, the HOMINID model can be seen to focus on the hyperlink:

- The Temporal Reference redefines the hyperlink's semantics to include the dimension of time.
- The RLS manages the hyperlink's integrity, ensuring it can be located across time and space regardless of how often it moves.
- The Oracle Server manages the informational integrity of the hyperlink, ensuring it can once again become an effective part of the web's navigation mechanism.

In this way, the HOMINID model provides a new model for managing information flow on the web that has been designed to work according to the web's open philosophy, and within its existing architecture. As such, the HOMINID model has been designed to work *with* the nature of information flow on the web, rather than against it, and so should stand a better chance of adoption than existing solutions.

This chapter, however, has only presented a conceptual overview of the HOMINID model. As such, the next chapter presents its design specification in detail, which defines how the model can be deployed on the web without breaking its existing architecture. The design is verified in chapter 6 by a prototype of the RLS, which has been developed and measured as part of this research programme.

5. Architectural Design of the HOMINID Model

The HOMINID model provides a new model for managing information on the web. This chapter presents in detail the model's architectural design, which enables it to be integrated into the web without breaking the web's existing architecture. The designs of the RLS, temporal references, and the Oracle Server are discussed, together with the Request Router, a novel object that mediates between the web's existing architecture and the new entities of the HOMINID model.

5.1 Introduction

The core components of the HOMINID model are the temporal reference, the RLS and the Oracle Server. These components must be deployed on the web as distributed systems that are fully backwards compatible with the web's existing architecture. To facilitate this, a new system of mediation between the web and the components of the HOMINID model has been designed, called Request Routing. This is a generic method of locating specific nodes in a distributed system in a scalable, transparent way. This chapter describes Request Routing in depth, and shows how it is used in the design of the RLS and the Oracle Server.

5.2 Designing the Resource Locator Service

The RLS is perhaps the most important part of the HOMINID model, as it provides an elegant solution to link rot and the shrinking namespace, while providing the web with a means of archiving its old ideas. The overall design of the RLS is described in detail in this section, while the design of its resource migration functionality is described in chapter 6.

5.2.1 The Scope of the Resource Locator Service

In order for the RLS to provide its services, a resource must first be registered with the RLS, in a similar way to the registration process required by the DNS. Registration with the RLS involves the resource owner submitting information about the resource (such as file-size, content type, etc.), which is used by the destination server during automatic migration to decide on the suitability of the resource according to the server's hosting policy (see section 6.2). The information is also submitted to the Oracle Server as characteristic infons (see section 5.5.3.2). Registration also identifies those resources that need managing without

requiring the RLS to crawl the web. In addition, it gives the resource owner the choice of whether or not to use the RLS, as it is designed to co-exist with the DNS, which can still be used if required. In this way, the scope of the RLS is limited to those resources that have been registered with it, but the design is such that all resources can still be accessed by all clients, regardless of whether they use the RLS or the DNS.

5.2.2 Selecting the Approach to Resource Migration

Resource migration is a core feature of the RLS, and the approach adopted for its implementation will determine the RLS's architectural design. Recall that section 3.2.4.1 discussed the five different approaches to resource migration. However, the requirements of the RLS, including an unrestricted namespace, temporal references, and integration with the web's existing architecture, preclude some approaches, as they are unable to meet the constraints placed on its design. For example:

- *the Callback approach* operates at the level of the hyperlink, by attempting to update a hyperlink within its containing web page so that it references the resource's new location. However, for the RLS to operate effectively using this approach, all hyperlinks referencing a resource would have to be updated whenever the resource migrates or its content changes. As such, this approach cannot scale, as it requires too many hyperlinks being updated too frequently right across the web. Thus, the Callback approach to migration cannot be used with the RLS.

- *the Chain approach* would require the software in existing servers to be updated, and so would not be backwards-compatible.
- *the Search approach* advocates searching the whole web for every resource each time the user browses to a new resource. This approach does not scale on today's web (Ingham et al., 1996), but the situation would be made worse if it had to manage temporal as well as spatial references.
- *the lecturing approach* cannot work while there is no gatekeeper or entity that can enforce the lecture.

This leaves the name server approach as the only suitable approach to resource migration for the design of the RLS. This requires the RLS to be configured as a network of name servers, each of which must resolve the name of a resource onto its location whenever a client application requires this service. The RLS's name server is termed a *Locator*, which maps a resource's name onto its location, and maintains the integrity of this mapping by persisting the name and updating the location whenever the resource moves.

5.2.3 *Removing the Namespace Constraints*

By adopting the name server approach, the RLS can be seen as a distributed database, with each Locator acting as a node in the database, storing a subset of the total information in the system.

In order to operate according to the requirements of the HOMINID model, this information must comprise:

- the persistent name of the resource, the syntax of which cannot be constrained.
- the current location of the resource, defined according to whatever identifier can be used by a client system to uniquely locate it.
- the time of the resource's creation.

The persistent name will be used by hyperlinks and users to reference the resource, whereas the dynamic location will be used purely by the RLS to locate the resource. The namespace of both name and location should be unconstrained, enabling many different types of naming schemes to be mapped onto different types of location addresses, such as IP addresses, phone numbers, or the co-ordinates of the Global Positioning System (GPS). The RLS is not required to understand the syntax of either the name or the location, but simply to return the location when given a name. In this way, the RLS provides a flexible resource migration mechanism that prevents link rot for all resources that it manages, ensuring the preservation of those resources' hyperlinks throughout the lifetime of the resources. A full description of the resource migration mechanism, including a novel Resource Migration Protocol, is provided in the following chapter.

Note that the semantics of the name under the RLS are similar to that of the URN. However, although the RLS supports the URN namespace, the persistence of a resource's name does not necessarily make it a URN. Berners-Lee et al. (1998) defines a URN as a persistent identifier for a resource, which must exist *beyond* the lifetime of the resource, and which is constrained by the URN syntax defined in RFC 1737 (Sollins and Masinter, 1994). In contrast, the resource name used by the RLS must only persist for as long as the resource still exists, and its syntax is completely unconstrained, so long as it is unique within the RLS.

Also note that for the purposes of this chapter and the following one, the Locator will be described as storing a resource's name/location mapping; that is, the record within the Locator's database that maps a resource's name onto its location. This simplifies future discussions when describing the RLS's architecture, deliberately ignoring the time of the resource's creation for clarity. The temporal aspect of the Locator is defined in full in section 5.4. To simplify the discussion further, although a name formed using any syntax can be mapped onto a location formed using any syntax, it will be assumed that the format of a resource's name will be a standard URL or a temporal URL, and the format of its corresponding location will be a standard URL.

5.2.4 Defining the Locator's Client-Side Interface

As a Locator acts as a name server, it will be queried by existing web clients for the location of a specific resource. In order to maintain backwards-compatibility with these clients, the Locator must interface with them using an existing protocol, and so remain transparent to them. This constrains the Locator to use either HTTP or the DNS's message format (see

Mockapetris, 1987b) as its functional interface to web clients, as these are the only protocols that all clients use when requesting a web resource. Of the two, HTTP has been chosen, as it already has a redirect mechanism (see Fielding et al., 1999) that can be used by a Locator to redirect a client to a resource's current location. Although HTTP's redirect mechanism is quite heavyweight in its operation, it is the only option that ensures the Locator will work with all existing web clients. Performance measurements of the overhead that the Locator introduces to the web are provided in section 6.4.5.

Figure 12 shows a high-level overview of a client interfacing with the RLS, with HTTP being used as an interface onto the Locators.

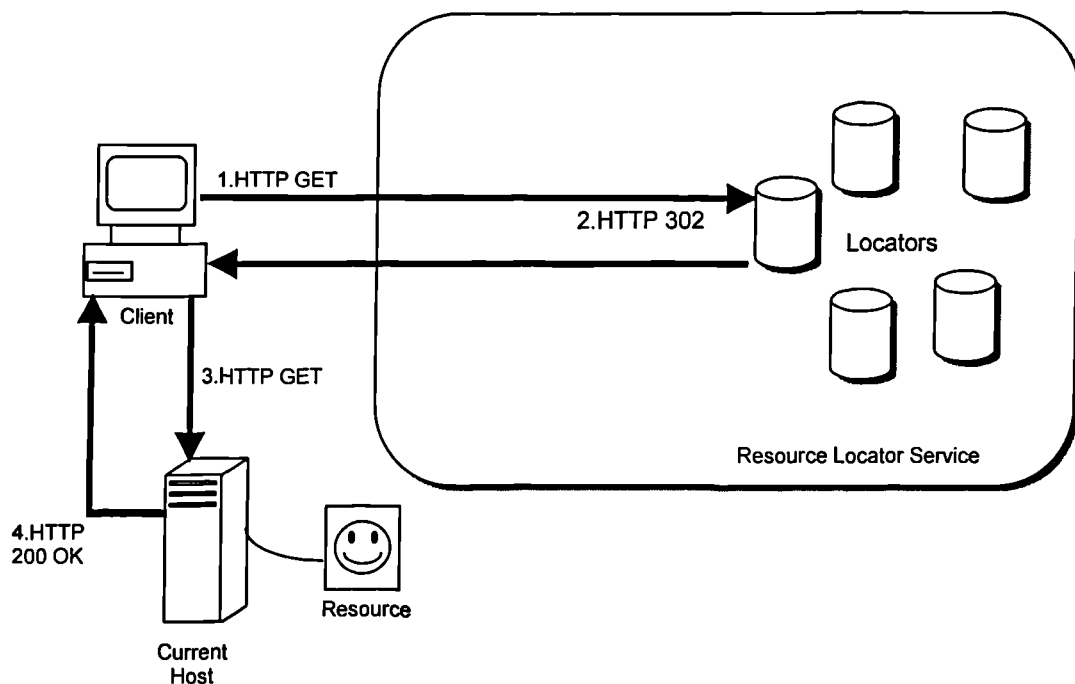


Figure 12 - A High-Level Overview of the RLS

The diagram shows a client sending a standard HTTP GET message to a Locator as if the Locator was the server hosting the requested resource. The Locator examines its database, retrieves the location of the resource, and responds with a standard HTTP *302 Found* response message, which informs the client that the resource has moved and provides the new location using the *HTTP Location* header (Fielding et al., 1999). If the Locator has not got a record of the resource's name, then it must return a standard HTTP *Error 404 Not Found* response message.

5.2.5 Missing Mediation

Figure 12 is a high level representation of the RLS, and depicts the client querying an appropriate Locator. However, in practice, a client must be made aware of exactly which Locator contains the required name/location mapping, but in a way that does not require the client or the hosting server to be altered. As such, some form of mediation is required between the client and the RLS that can transparently route the client's request to the appropriate Locator, without requiring any modifications in the client or server.

This is difficult to achieve, however, as the constraints imposed on the RLS directly conflict with its distributed nature. For example, some distributed systems, such as the DNS or directory services, use the structure of the namespace itself to identify the correct node, but the RLS cannot, as the namespace must be left completely unconstrained. The alternative is a flat architectural configuration, with nodes arranged as peers, and the namespace left unconstrained, but this requires the search approach to be used, which will not scale to a system the size of the web (Ingham et al., 1996).

As such, in order for a client to locate the correct Locator in the RLS, a new approach to mediation is required that:

- does not use a flooding algorithm;
- can locate the node with the required information as easily as that of a hierarchical architecture;
- leaves the namespace as unconstrained as that of a flat architecture;
- does not impact the existing web architecture.

This new approach has formed a major part of the work conducted for this research programme, and is presented in the following section.

5.3 Request Routing: Novel Mediation Between the Web and a Distributed System

To meet the constraints imposed upon the RLS, a novel solution has been developed that mediates between the web and any new distributed system that wishes to interface with it. Specifically, a novel node location system has been developed as part of this research programme, which places no constraints on the namespace, does not waste bandwidth, and which leaves each Locator as an independent node that has no knowledge of any of the other Locators in the RLS. The node location system is called *Request Routing*, which uses a

Request Router to provide transparent, scalable mediation between the web and the RLS through the use of a *hash routing* algorithm (Ross, 1997).

Hash routing is an extension of the hash function, which is a common method of searching for information in a large database. The hash function works by scrambling some aspect of the database key, and using this partial information to search for the data required (Knuth, 1998). Hash routing extends this concept, by efficiently mapping a string (in this case a resource's name) onto a specific server in a distributed system, while ensuring a uniform distribution of resources across the servers in the system (Thaler and Ravishankar, 1998).

Specifically, a hash routing algorithm takes a string and maps it onto a hash space. The hash space is partitioned such that the string is mapped to one and only one node in a distributed system (Ross, 1997, Thaler and Ravishankar, 1998). Using a hash routing algorithm as the basis for locating nodes in the RLS, therefore, enables any string to deterministically identify the Locator that contains the required name/location mapping. As the Locator is a database, it can be defined to store any type of information, and so the resource's location can be defined as any string. Thus, the hash routing algorithm solves the problem of how to use an unconstrained namespace for both a resource's name and location, while efficiently locating the correct Locator without flooding the system.

5.3.1 The CARP Hash Routing Algorithm

The Request Router (RR) uses the same hash routing algorithm as the Cache Array Routing Protocol (CARP) (Valloppillil and Ross, 1998), which uses it to map a URL to a specific

cache in a distributed caching system. CARP uses hash routing to distribute the resource load across all caches in a CARP system, such that resources within the cache are distributed evenly across an array of machines. The algorithm has been adapted for use in the RLS, however, and has two key differences:

- The CARP approach takes a name and returns a (copy of a) resource, whereas the RLS returns the actual resource's location.
- The CARP approach uses internal lookup tables to keep track of the other machines in the cache array, and each node must intermittently ping its neighbours to determine the size of the array and its operational state. In contrast, the RLS adopts a novel approach to node management, with each Locator being completely isolated from its peers.

5.3.2 How the CARP Hash Routing Algorithm works

The CARP protocol is designed for an array of caching servers. As such, the array comprises a network of distributed caching nodes, and the CARP protocol is used to identify which one hosts a specific resource. CARP works by mapping the URLs of resources that need to be cached onto a partitioned hash space, with each set in a partition being associated with one caching node (Ross, 1997). When a resource is required, the algorithm deterministically identifies the node as follows:

- The URL of the resource is hashed.
- The URLs of each of the caches in the array are also hashed in turn, with a weighting factor being applied that is set according to the physical characteristics of each node (see below).
- The hash value of the resource and the hash values of the nodes are XORed together, producing a score for each resource-URL-hash/cache-node-hash combination.
- The cache node whose resource-URL-hash/cache-node-hash combination scores highest is the one that hosts the resource.

Thus, given only the name of the resource and the names of all the machines in the array, the exact machine that holds the resource is uniquely and deterministically found. The resources are distributed uniformly across the system, but the weighting factor can be used to skew the distribution such that those nodes with a higher performance can receive more of the resources.

5.3.3 Adapting the CARP Hash Routing Algorithm for the RLS

Although highly effective in large cache arrays, the CARP protocol has been adapted for use in the Request Router in order to better meet the needs of the RLS. Specifically, each node in a CARP system keeps a list of the URLs of all the caches in the system, and this causes a degree of network overhead. When applied to the RLS, however, every RR would need to

know the URL of every Locator in the RLS, and periodically check for system configuration changes. This would create an unacceptable increase in network overhead, and would limit the types of device that could use the RR to those that could store and maintain the large lists of Locators that would be required.

The RLS avoids this limitation, however, by removing the weighting factor from the algorithm, and leaving the namespace of the resources open while restricting the namespace of the Locators. As such, a *URL pattern* is defined, which all Locators must use for their own name. The pattern encapsulates a number, which can be thought of as that Locator's identity number. Each number must be unique in the system, and all numbers must be sequentially ordered, starting from 0. An example URL pattern is:

http://www.nodeX.Locator.net/

where X is a marker for where the Locator's number should be. So for example, the first three nodes in the system (assuming a zero-indexing system) would be:

http://www.node0.Locator.net/

http://www.node1.Locator.net/

http://www.node2.Locator.net/

Effectively, the URL pattern of the Locators acts as a *well-known* URL in a similar way to the well-known ports defined for TCP applications. Note that the URL pattern must be sequential,

and there can be no gaps in the sequence. The URLs of a complete sequence of nodes, each of which has a URL that corresponds to the URL pattern, is therefore known as a *URL sequence*. In this way, the URLs of the Locators themselves become deterministic.

5.3.3.1 Updating the Request Router

Simple hash routing schemes are brittle, such that the addition or removal of a node in the system will re-map nearly all of the URLs onto different nodes (Ross, 1997). Robust hash routing algorithms, such as CARP and its variant used in the RR, overcome this problem by using the name of the resource and the name of the node together, which results in the re-mapping of only $1/n$ (where n = the number of machines in the new system configuration) of the URLs in the system (Ross (1997), Thaler and Ravishankar (1998)). As such, the number of name/location mappings in the RLS that must be moved *decreases* with the number of nodes in the system. The RLS provides a mechanism for automated node removal and addition, which is described in section 6.3.

If a Request Router is unaware of a change in the system's configuration, then $1/n$ of its requests will go to the wrong Locator. However, the RR does not need to be synchronized with the configuration of the RLS, as the deterministic nature of the URL sequence enables it to detect any change automatically. Specifically, once the RR has the URL pattern for the RLS, it is a trivial matter for it to iterate along the resulting URL sequence, querying the existence of nodes at each point in the sequence. If a Locator fails to respond, then the RR has found the limit of the sequence (see Figure 13).

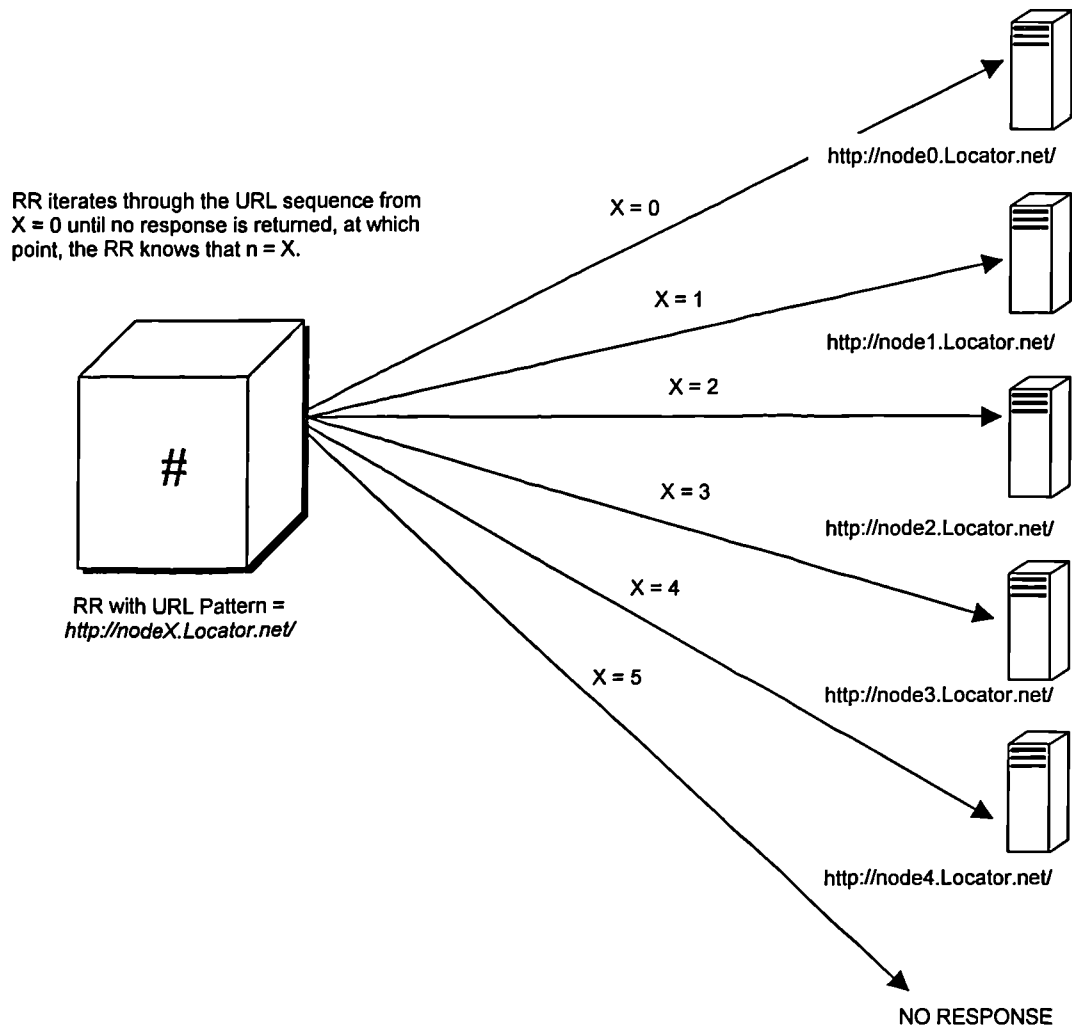


Figure 13 - How the RR updates itself

Thus, if a Locator cannot find a resource, the RR can simply query the existence of the Locators that have a node number that matches this limit (in case a Locator has been removed), or is one greater (in case a Locator has been added). If the limit remains unchanged, then the RR knows that the resource is unregistered with the RLS; otherwise, the RR simply rehashes the resource's name using the updated value, and sends the request to the newly calculated Locator. In this way, the RR is completely decoupled from the configuration of the RLS, and so any change in the configuration of the system does not result in a flood of

update messages. Furthermore, the only information that the RR needs to store about the configuration of the system is the URL pattern and the number of nodes.

5.3.3.2 Backwards Compatibility

For the RLS to integrate into the web's current architecture, it must be backwards-compatible with all of the entities that currently use the web. In this, the RLS is completely different from all other resource migration mechanisms, as the RR is decoupled from the RLS, and needs only minimal information in order to function, enabling it to be deployed virtually anywhere on the web. For example, it can be:

- *embedded into a HTML document as a Java applet, ActiveX control or even script.*

When the user clicks on a hyperlink, the click event can be captured by the embedded RR, the hash routing operation performed, and the location of the Locator discovered. Thus, the node location process occurs within the HTML page itself. This ensures total transparency and maximum backwards compatibility, but permits only HTML documents to use the RLS;

- *built into a browser.*

The browser automatically locates the appropriate Locator, allowing all servers to be unaware of the RLS, but requiring the client to be modified;

- *designed as a browser plug-in.*

The browser is *extended* rather than redesigned, with the RR being downloaded by the user when required. This provides seamless evolution, and a solution that is more backwards-compatible than the previous example. Again, all servers are unaware of the RLS;

- *built into a server, or added as a server module.*

The RR can be deployed on the server, which can perform the hash routing algorithm for each request it receives. This allows all browsers to be unaware of the RLS, and gives server owners the choice of whether to use the RLS or not;

- *embedded within a proxy server or a reverse proxy server.*

The proxy server intercepts the request, and routes it to the appropriate Locator. This requires reconfiguration rather than redevelopment, allowing all browsers, servers and resources to be unaware of the RLS;

- *designed into a layer 4 switch or policy based router.*

The switch or router contains the RR, transparently routing the request to the appropriate Locator without client or server knowing. This provides total transparency and maximum backwards compatibility.

This flexibility enables the RR to be integrated into the web at the position where it is required. In this way, the number of resources registered with the RLS can grow over time as more people decide they wish to use its services. As such, the adoption of the RLS is designed to be evolutionary, rather than revolutionary, proceeding in a distributed way across different sectors of the web, as its services become useful to different types of user. For example, to begin with, small numbers of web authors may embed a RR within a HTML document. After a short period, server owners may decide to embed a RR into their servers in order to use the RLS without affecting the clients. From this, plug-ins can be made available for existing browsers, allowing resources to be located directly in the browser, via both the DNS and the RLS. Once a reasonable number of people use the RLS, Internet Service Providers can embed a RR into their proxy servers. Eventually, the RLS will reach a critical mass of users, whereby a RR will become an integral part of a browser and server, and thus part of the web itself. In this way, the RLS's database becomes populated over time by resource owners who choose to register their resources with it. As such, the database does not need to be initialised, and because it freely co-exists with the DNS, does not prevent non-registered resources from being accessed. In this way, the RR represents a novel solution to integrating new distributed systems into the web's existing architecture.

5.3.4 How the Hash Routing Algorithm Works in the RLS

Figure 14 shows the architecture of the RLS. The figure depicts a client that wishes to locate the resource whose name is *http://www.anyserver.com/img.gif*. The RR is represented abstractly as a box, to show that it can be positioned anywhere in the location process. For the

purposes of this example, however, assume that it is in a proxy server that the client is connected to.

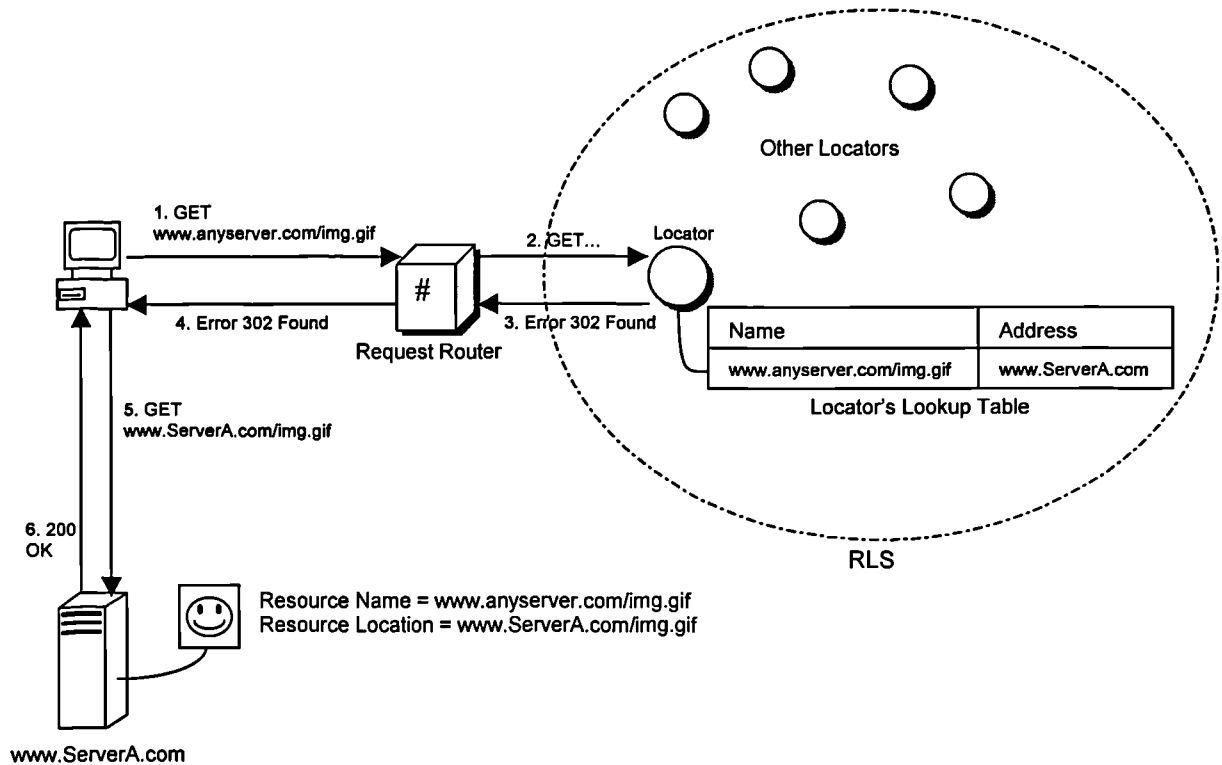


Figure 14 - The Architecture of the Resource Locator Service

In order to locate the resource, the client sends a standard HTTP GET request (Fielding et al., 1999), as it would do normally if retrieving the resource without the aid of the RLS. However, the proxy server intercepts the call, and passes the URL to the hash routing algorithm. The algorithm begins by calculating a hash value for the URL (*URL_Hash*), using the CARP algorithm developed by Valloppillil and Ross (1998):

```
For (each char in URL): URL_Hash += _rotl(URL_Hash, 19) + char;
```

Once the URL_Hash value is found, the hash value must be found for each of the Locators in the RLS (*Locator_Hash*). Again, using the CARP algorithm developed by Valloppillil and Ross (1998), the algorithm does this as follows:

```
Locator_Hash += Locator_Hash * 0x62531965;  
Locator_Hash = _rotl (Locator_Hash, 21);
```

For each Locator_Hash value calculated, the algorithm must note which Locator's URL it was derived from. Once each Locator_Hash value has been calculated, the algorithm proceeds to combine each value with that calculated for URL_Hash, as follows (see Valloppillil and Ross (1998)):

```
Combined_Hash = (URL_Hash ^ Locator_Hash);  
Combined_Hash += Combined_Hash * 0x62531965;  
Combined_Hash = _rotl (Combined_Hash, 21);
```

Again, for each Combined_Hash value calculated, the algorithm must note which Locator's URL it was derived from. The algorithm will then be left with a series of values for Combined_Hash (one for each Locator in the RLS), each matched with the URL of the Locator from which it was derived. The algorithm can then simply select the Locator URL matched to the highest scoring Combined_Hash value to find the Locator that stores the URL's name/location mapping.

The hash routing algorithm returns with the correct Locator's URL, which the proxy server uses to locate the correct server to pass the original HTTP GET request onto. The Locator then retrieves the URL *http://www.anyserver.com/img.gif* from the HTTP GET request, and uses it as the key into its database, where it retrieves the location of the resource. The location is passed back to the proxy encoded in an HTTP 302 redirect response message, which is returned without change to the client that made the original request. As such, from the client's perspective, a standard HTTP request has been met with a standard HTTP redirect response, and the resource can be located at the location specified in the response message's *location* header according to the standard HTTP 1.1 protocol. In this way, the RLS is completely transparent, and so is able to effectively integrate into the existing web's architecture without requiring any modifications of existing web entities.

5.3.5 The Design of the Request Router

The Request Router is a generic entity that encapsulates the hash routing algorithm, and uses it to uniquely identify a node in a distributed system when given a string identifier. As such, the RR can be used with any distributed system whose nodes comply with its back-end interface, and which uses HTTP to communicate. For the purposes of this discussion, however, the system is assumed to be the RLS, the node is assumed to be a Locator within the RLS, and the identifier is the URL of a resource.

The Request Router must accept any string as input, perform the hash routing function on it, and output the URL of the correct Locator in the RLS. More specifically, the RR must provide two interfaces: a function-oriented front-side interface, which clients of the RLS use

to query a Locator with; and an HTTP-oriented back-end interface, which is used to interface directly with the Locators via HTTP. Using HTTP in this way enables the RR to integrate with the web seamlessly, and to contact a Locator wherever the RR is deployed. In contrast, the front-side interface provides a standard API, which a client application must be adapted to if it wishes to use it. The client application can then provide a web-specific wrapper around this API, which hides the RR from the web entities that use it to maintain backwards-compatibility (for example, a proxy server provides a HTTP wrapper around the RR to hide it from a browser).

If the RR is asked to query the RLS for a resource that is not registered, the RLS will return an *Error 404 Not Found* message. It will not attempt to locate the resource using the DNS, as this may not be appropriate in all cases. For example, a server hosting a RR may have registered all of its resources with the RLS, and so an *Error 404* from the RLS means that the resource does not exist, not just that it is not registered. As such, it would serve no purpose for the RLS to contact the DNS in this situation, and so the RLS only manages its own *registered* resources, leaving client applications to determine what to do with those that are unregistered.

In addition, a RR may receive an *Error 404* message from a Locator because the configuration of the system has changed, and the name/location mapping has moved to a different Locator. In this case, the DNS still has no record of the resource, but the RLS has. As such, the RR should update its view of the system configuration (i.e. it should perform an *automatic update*), but only if instructed to do so by the client application. If not, the RR should simply pass the *Error 404* back to the client application, and let it decide what should be done.

5.3.5.1 The Request Router's Interfaces

The RR's client-side interface has two functions that are used to identify the correct Locator (or, indeed, a node in an equivalent distributed system). The first, *RouteRequest()*, takes the name of the resource, and returns the appropriate Locator URL with the resource's name appended onto it as a Query String (e.g. *http://www.node1.Locator.net/query?resourcename=http://www.asever.com/aresource.htm*). This URL can then be sent directly to the appropriate Locator without the need for adding any new HTTP headers. The Locator will return a HTTP 302 *Found* response message, which will be acted upon by the browser as per standard HTTP conventions (see Fielding et al., 1999), and the resource successfully located. In this way, script languages such as JavaScript, which cannot ordinarily alter HTTP messages, can use the RLS, allowing a HTML document to locate migrated resources on a browser that has no knowledge of it (see Figure 15 for an example). Note that the *RouteRequest()* function will force the RR to perform an automatic update to check the system's configuration if an *Error 404* is returned. Thus, if the function returns an empty string, the client knows that the resource is not registered with the system.

```
// Assume RR is a pre-instantiated ActiveX RequestRouter, and that
// sURL is a resource whose location is known by the RLS
Function NavigateToResource(sURL)
{
    var sLocatorURL;

    //Get the URL of the correct Locator
    sLocatorURL = RR.RouteRequest(sURL);

    //sLocatorURL can be used directly by JavaScript to
    //navigate to the appropriate resource
    window.open(sLocatorURL);
}
```

Figure 15 - Sample JavaScript function showing a RR embedded in a HTML Page

The second function, *GetNodeByName()*, takes the name of a resource (which can be any string) and returns the URL of the appropriate Locator. This function only identifies the Locator, returning its location as a URL; it does not append the resource's name onto the Locator's URL. To retrieve the location of the resource, the Locator's URL can be used by the client application in a subsequent HTTP GET request message, together with a new HTTP request header called *resource-name*, which has the name of the resource itself as its value (this forms part of the RR's back-end interface, and must be implemented by all nodes in the distributed system that communicate with the RR). The Locator will then return a HTTP 302 *Found* response message, with the URL of the resource contained in the message's *location* header (Fielding et al., 1999). Note that the Locator requires *resource-name* to be a new request header because the existing headers in HTTP have inappropriate semantics. For example, the *resource-URI* header of the GET method specifies the URL of the Locator, not the web resource; the *host* header specifies the web server, not the resource; and the *location* header and *ETag* header are response headers only (Fielding et al., 1999).

The RR also has functions that enable the URL pattern to be changed (thus allowing it to interface with other distributed system on the web), and a function called *Update()*, which enables it to determine the number of Locators in a network by performing an automatic update.

Some clients, such as mobile phones, may have very limited processing abilities, and may not be able to perform the hash routing function required. As such, part of the RR's back-end interface includes a new HTTP request header *authoritative-lookup*, which is defined using a

Boolean value (default is false), and which forces the Locator that receives the request to perform the hash routing function itself if it cannot locate the resource in its own database. In this way, the client can send the request to an arbitrary Locator, and have the RLS itself locate the correct Locator, enabling the client to have no direct interaction with the RR at all. However, due to the performance overhead this places on the Locator, this functionality should be restricted as much as possible.

5.3.6 Scalability

In order to work within the web, the design of the RLS must be fully scalable in a number of areas. The following sub-sections describe the scalability of the RLS, while real world measurements are presented in section 6.4.5, which have been taken from an instrumented prototype of the RR that has been developed as part of the research programme.

5.3.6.1 Network Overhead

Hash routing is a very fast algorithm for locating a node in a distributed system, providing a deterministic request resolution path through an array of machines, which results in locating a specific node in a single hop (Microsoft, 1997b). As such, the network overhead introduced by the RLS for both a successful and an unsuccessful resolution operation is *always* two additional HTTP messages (either a GET and an *Error 302 Found* response, or a GET and an *Error 404 Not Found* response).

If the RLS cannot find the resource, then a client application may contact the DNS if required, and if this is successful, the round-trip time to the RLS via the RR has been wasted. If, however, the RLS is completely integrated into the web, such that the DNS is not used to find

the locations of resources, then all resources will be registered, and an *Error 404* means that the resource does not exist on the web, not just in the RLS. As such, there will be no added overhead, as the resource is unattainable.

5.3.6.2 CPU Overhead

The design of the RLS is such that the network overhead is constant, regardless of how many Locators are in the system, whereas the CPU overhead required by the RR scales linearly with respect to the number of Locators. As such, the scalability of the design is constrained more by CPU overhead than network overhead.

The linear scaling of the RR results from the hash function being used to distribute a set of records across many Locators, rather than to generate a unique value each time it is used, and so it does not have to worry about managing collisions, as the same result (i.e. the identified Locator) can be used many times for different resource names. The function distributes the records by hashing the URL of each Locator in the system, and as the time taken to hash each URL is virtually uniform (dependent solely upon the number of characters in the URLs that are hashed), the CPU overhead increases linearly with respect to the number of URLs (and thus Locators) it must hash.

5.3.6.3 Scalability of the Overall Design

In terms of growth, the hash routing algorithm can scale to over 4 billion ($2^{32} = 4,294,967,296$) Locators, performing single-hop resolution throughout (Microsoft, 1997b). Assuming each Locator can store the names and location of 1 million resources (which, assuming the name and location each use a URL that averages 50 characters, will require a database only 100 MB

in size), today's web, with over 1 billion documents, would need the deployment of 1,000 Locators to fully manage all resources. Alternative combinations of records-per-Locator against number of Locators can be tested to optimise the configuration, but network overhead is unchanged regardless of the number of Locators. This makes the system scalable up to 2^{32} Locators, or 4 quadrillion managed resources. However, the CPU time taken to perform the hash routing algorithm limits the practical number of Locators that can be used. Section 6.4.5 discusses this in more detail, and provides real world measurements of the overheads that are involved.

5.3.7 Resilience

Because of the system's reliance on the URL sequence, it is not resilient to node failure. Should a Locator fail, not only will its records not be available, but any RR that performs an automatic update during the failure will calculate the wrong number of nodes in the system, and will map most URLs onto the wrong Locator. However, the disruption can be limited if the RR continues to check for the existence of nodes beyond that at which no response is received, effectively enabling it to jump any holes in the URL sequence. Although the RR will still not be able to access the records in the failed Locator, it will at least know the correct configuration of the system, and so all other records will be available. In addition, the system's resilience is actually better than that of the DNS, which, due to its hierarchical structure, has a single point of failure (the root node). The reliability of the DNS comes from the introduction of redundancy into the system, with distributed servers clustered to provide a single fault-tolerant node. As such, future work will look at introducing redundancy into the design of the RLS, either by clustering several servers to provide a more fault-tolerant Locator

design, or by using a *duplicated* hash routing algorithm, such as that proposed by Kawai et al. (2000). Duplicated hash routing uses two hash routing functions and two cloned systems, one of which is a secondary system that acts as a backup in the event of a node in the primary system failing. However, the benefits of this algorithm need to be determined, as although the reliability of the system is improved, the size and complexity are increased.

5.3.8 Impact of the Resource Locator Service on existing Web mechanisms

The RLS has been designed to be backwards compatible with the existing web architecture. The RR ensures that it can be integrated into the web without affecting either clients or servers, but there are other systems, such as caching servers, that also need to be considered if the RLS is to be effectively deployed. This section considers the impact of the RLS on these systems.

5.3.8.1 Impact on Caching Servers

Caches are an integral part of the web, and help to speed up resource delivery dramatically. Caches exist at all levels of the system, from an enterprise level, though ISP level, and up to country level, with massive caches storing resources that are hosted outside of a country in order to minimize the traffic that passes across expensive long distance lines. In order to work effectively within the web, the RLS should not have a negative impact on such systems.

When a cache receives a request for a resource for the first time, it forwards the request onto the appropriate origin server, and stores the returned response before passing it back to the requesting client. Subsequent requests for the same resource are then served directly by the cache providing the stored resource is still fresh. With the RLS being used, the initial request

by the client will be to the appropriate Locator, and the response will be an Error 302 Found message. HTTP 1.1 defines that the cache must not store this response message, unless explicitly instructed by the origin server (in this case, the Locator) (Fielding et al., 1999). The Locator will not instruct the cache to store this response message, and so all subsequent requests for the same resource will always be passed through the cache and onto the appropriate Locator. As such, the amount of traffic between the cache and the Locator will be more than would have existed between the cache and the origin server. However, the traffic involved will only be request and response messages, and not the actual resource itself. As such, the extra traffic incurred should be minimal.

Upon receiving the 302 Found response message, the client will issue another request for the resource, to the server at the specified location. The cache will receive this message as a separate GET request, and will retrieve the resource from the server. The cache will then store the resource before passing it onto the client. All future requests for this resource can then be served by the cache. As such, the RLS does not increase the number of cache misses for the resource.

The impact of the RLS on caches, therefore, is to create an overhead of only two extra HTTP messages (i.e. those between the cache and the Locator) per request, as the cache will serve the subsequent redirected request. This overhead is the same as that caused by the RLS without a cache being used, and so the overall impact on a cache is negligible.

5.3.8.2 Impact on History and Bookmark Mechanisms

The RLS relies on the HTTP redirect mechanism to serve the requested page. However, a browser will navigate to the new location after receiving a redirection command, and use the new URL in its History and Bookmark mechanisms. Thus, when a user navigates to a resource that has migrated, it is the resource's transient location that will be stored by the browser's history and bookmark mechanisms, rather than its persistent location.

This situation can be avoided by proxy servers that contain the RR and retrieve the resource from its location on behalf of the client. The client would then never receive the redirect response message from the Locator, and so would use the persistent name of the resource in its bookmark and history mechanisms, rather than the resource's location. Alternatively, browser plug-ins that contain the RR can be made to transparently alter the mechanics of the history and bookmark mechanisms, such that the name and not the location is stored. As such, although the RLS does have a small impact on these mechanisms, it can easily be overcome.

5.4 Temporal References

The temporal reference is a core part of the HOMINID model, and can be integrated into the web through the RLS. As the RLS can support any string as the name or location of a resource, it can support the syntax of a temporal reference just as easily as it can a URL. However, care must be taken in the design of the temporal reference's syntax to ensure that it can safely co-exist with the URL to maintain backwards-compatibility. As such, two different versions of the temporal reference have been defined:

- A completely new *Temporal URL* scheme, which provides a long-term architectural solution, but which currently only works with the RLS.
- A *URL extension*, which provides a backwards-compatible short-term solution, but is less elegant.

5.4.1 *The URL Extension*

The *URL extension* version of the temporal reference comprises a URL with a *timecreated* temporal component appended as a Query String, which allows existing URLs to be used as temporal references. For example, the extended URL:

<http://www.asever.com/index.htm?timecreated=Sun,%2006%20Nov%201994>

has *<http://www.someserver.com/index.htm>* as its location component, followed by ? as a separator, and *timecreated=Sun,%2006%20Nov%201994* as its temporal component. Note that %20 is the URL encoding for whitespace, and that the time and date are formatted according to RFC 1123 (Braden, 1989). Query Strings are an integral component of the URL specification, and are used to pass parameters to servers (Berners-Lee et al. 1994). However, servers must ignore parameters they do not need to use, and so adding a *timecreated* parameter to a URL's Query String will enable existing hyperlinks to become temporal references without requiring the modification of browsers or servers.

Those URLs that exist without a temporal component are re-defined as *current URLs*; that is, they represent the most current version of a resource. Once the content changes, the new resource with the new content is assigned the current URL without the *timecreated* QueryString, and the old resource with the original content is assigned the same URL, but with an appropriately formatted *timecreated* QueryString appended onto it. Note that for successful resolution of such extended URLs, the server must be able to determine which resource to serve according to the *timecreated* QueryString. However, all existing URLs can be treated as current URLs, without requiring any modifications to the server. Formally stated, a temporal URL extension can be defined as:

a standard URL with a temporal component encoded in its Query String using the timecreated parameter, and a corresponding value that must not exceed the current GMT time, and that must be encoded according to RFC 1123.

5.4.2 The Temporal URL Scheme

The new *Temporal URL scheme* is an architectural solution that conforms to the encoding rules defined in Berners-Lee et al. (1998), and encapsulates the same semantics of the URL, but with the addition of a temporal component. Specifically, the new scheme, called TURL (Temporal Uniform Resource Locator), has been defined as:

turl://authority/path;time-created?query

The *authority* component of the TURL is identical to that of the URL (i.e. the domain name of the hosting server). The *path* component, too, is identical to the URL, but with one exception:

a semi-colon separates the path that the server uses to locate the resource from the temporal information used to identify the time that the resource was created. The *query* component remains as it is defined for the URL, but the whitespace of the temporal component has been replaced with a dash (-) for clarity. Thus the temporal URL extension:

http://www.asever.com/index.htm?timecreated=Sun,%2006%20Nov%201994

can be re-written as a TURL as:

turl://www.asever.com/index.htm;Sun,06-Nov-1994

In addition, as HTTP essentially forms the interface between the RR and the Locator, it has had to be extended in order to map the temporal component of the TURL onto a HTTP header. HTTP's existing headers already encode temporal information, but they are largely used for caching, and are normally sent by the server rather than the client. For example, the *Last-modified* entity header is used to represent the time at which the resource was last modified (Fielding et al., 1999), which is another way of saying the time at which the resource was created. However, it can only be used by servers in a response message, and cannot be used by a client at all. Equally, the *Age* entity header, which provides the estimated age of the resource on the origin server (Fielding et al., 1999), is also a response header, only sent by a server (usually a caching proxy server). Alternatively, the *Date* header field is a general header, which can be used by both client and server, but only to represent the date and time at which the *message* was originated, not the resource (Fielding et al., 1999). Finally, the *ETag*

header could encode the temporal information, as it provides a means of encoding user-defined values, but it, too, is a response header (Fielding et al., 1999).

As such, rather than subtly altering the semantics of existing HTTP headers, the RLS uses a new general header, called *time-created*, which can be used by both client and server, and which defines the time at which the resource was created. The value of the new header must be formatted according to RFC 1123 (Braden, 1989), and it must map exactly onto the temporal component of the TURL. The new header provides the preferred means for querying a Locator according to a resource's time of creation, thus separating the temporal information from the resource's name. In this way, any appropriately specified namespace is able to become a temporal reference by mapping its temporal component onto this new HTTP header, enabling the RLS to retain its unconstrained namespace.

5.4.3 Defining the Scope of the Temporal Reference

A temporal reference supported by the RLS can enable one resource to persist across time, but not the resources behind any hyperlinks that might be embedded within it. For example, a HTML document registered with the RLS may contain several hyperlinks, but if the resources underlying the hyperlinks are not registered with the RLS, then they may not persist. The RLS preserves a resource's referencing hyperlinks, therefore, but cannot guarantee the integrity of the hyperlinks within the resource. As such, the RLS can only prevent link rot for those resources that it has been instructed to manage, and so web-wide link rot prevention can only be achieved if the RLS manages all web resources.

In addition, transient resources, such as dynamically created HTML documents, or streaming audio or video, are also not covered by the current design of the RLS and the temporal reference. This is because the semantics of the TURL simply extends those of the existing URL protocol to encompass time, rather than adding any new functionality, and an existing URL references the object that creates a dynamic resource or a multimedia stream, rather than the transient resource itself. For example, a URL might identify an application behind a CGI (Common Gateway Interface) gateway, which in response returns a dynamically generated HTML document, but it does not identify the HTML document. Similarly, temporal references may enable the *application* to persist (although their definition does not cover persisting the application's state, merely its existence as a discrete file), but they do not cover its output, unless it is explicitly saved as a permanent web resource and given its own (temporal) URL.

5.4.3.1 The URL Extension Versus the Temporal URL

Using the URL extension enables temporal references to be implemented immediately without any change to the web's architecture, and should be used when the RLS is first deployed on the web. However, the TURL provides a more long-term solution, and should be the preferred identifier once the RLS has become adopted as part of the web's architecture. Thus, new versions of browsers and servers should support both forms of identifier, while all Locators *must* support both identifiers. As an intermediate solution, a plug-in or ActiveX control can be developed that extends the functionality of existing browsers to enable them to support the TURL.

Note that some applications may wish the Locator to return a number of resources, whose *time-created* value lies between certain times. However, this will not be defined for this version of the Locator, as it introduces the scope for potential Denial of Service attacks, and extends the functionality of the Locator to include database querying. This would require additional work to avoid the security implications, and extra HTTP headers to enable the Locator to be queried. As such, this work is left to the client to do at this stage, but future work will examine the possibility of providing this feature.

5.5 *Designing the Oracle Server*

The role of the Oracle Server is to manage the informational integrity of the hyperlink, by providing universal access to characteristic and navigational infons about the resources on the web and the way in which they are used.

Due to time and resource constraints, the Oracle Server will not be defined in as much detail as the RLS. However, the design of the Oracle Server is similar in many ways to the RLS, and so can use many of the techniques that were used in the RLS's design. As such, the following sub-sections functionally define the Oracle Server, and describe how it can be implemented, but do not provide a full definition of its architectural design.

5.5.1 *The Oracle Server Network*

One Oracle Server cannot cope with managing meta-data for all the resources on the web, and so a distributed network of Oracle Servers, called the *Oracle Server Network* (OSN), is required.

The OSN also uses a Request Router for its node location system, with the URL pattern of:

`http://www.nodeX.OracleServer.net`

Using a Request Router enables the OSN to be queried with the URL that the user wishes to know more about. In this way, the URL (or other identifier⁶) is used by the user to navigate across the web, and by the RLS and the OSN as an index into their respective databases.

5.5.2 The Architecture of the Oracle Server Network

Figure 16 shows the architecture of the Oracle Server Network. The user clicks on a hyperlink to download a resource, causing the browser to send a HTTP GET request. In this example, the client is connected to a proxy server that contains a RR capable of routing into the OSN, but the RR can be located wherever there is appropriate functionality for interpreting the OSN's results. The proxy forwards the request onto the origin server and downloads the resource. However, rather than returning the resource back to the client, it extracts all the hyperlinks from it and submits each one to the OSN for a judgement on its quality and the strategy that it uses, according to the heuristics used by the OSN. If the verdict is OK then the proxy can change the colour of the hyperlink in the resource to green (for example); if the verdict is that the hyperlink is deceptive, the proxy can colour it red; and if the OSN has not got enough information to reach a verdict, the proxy can colour it amber (note that in this

⁶ Both the RLS and the OSN can support any type of resource name, regardless of syntax. However, for clarity, the term *URL* will be used in the discussion, as it represents the most prevalent and identifiable type of resource name currently on the web.

example, neither the client nor the origin server needs to be altered in order to present this information).

In Figure 16, for example, the hyperlinks *www.link1.com* and *www.link3.com* are safe; *www.link2.com* is deceptive; and *www.link4.com* is unknown. Note that the exact presentation of the information should be user-defined, and not dependent solely on colour. The OSN is designed as a system for judging the informational integrity of hyperlinks; how that information is presented is up to the client application.

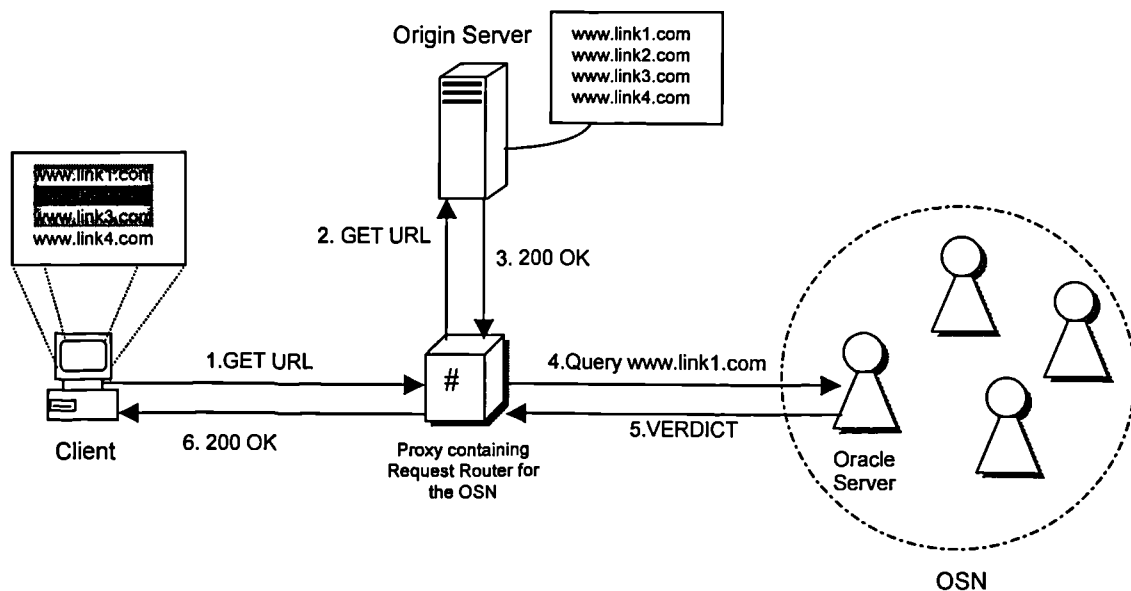


Figure 16 - The Architecture of the Oracle Server Network

5.5.3 Obtaining the Infons

The OSN obtains its navigational and characteristic infons from the RLS, with which it has been designed to work closely. As such, the OSN can manage only those resources that are registered with the RLS, as it cannot obtain the required infons from anywhere else.

5.5.3.1 Navigational Infons

The navigational infons come directly from the RLS each time a user downloads a resource. Whenever a user clicks on a hyperlink to a registered resource, the RLS is queried for the resource's location. This query represents a navigational infon, as it implicitly registers a hit on the resource. If the RLS manages all of the web's resources, then it can provide the OSN with all of the navigational infons across the web.

Figure 17 shows how the navigational infons are obtained from the user's browsing sessions, as the URL of the hyperlink is passed from the browser to the RLS, which passes it onto the OSN before returning the resource's location. In this way, the OSN updates its hit statistics for a URL every time the user clicks on a hyperlink.

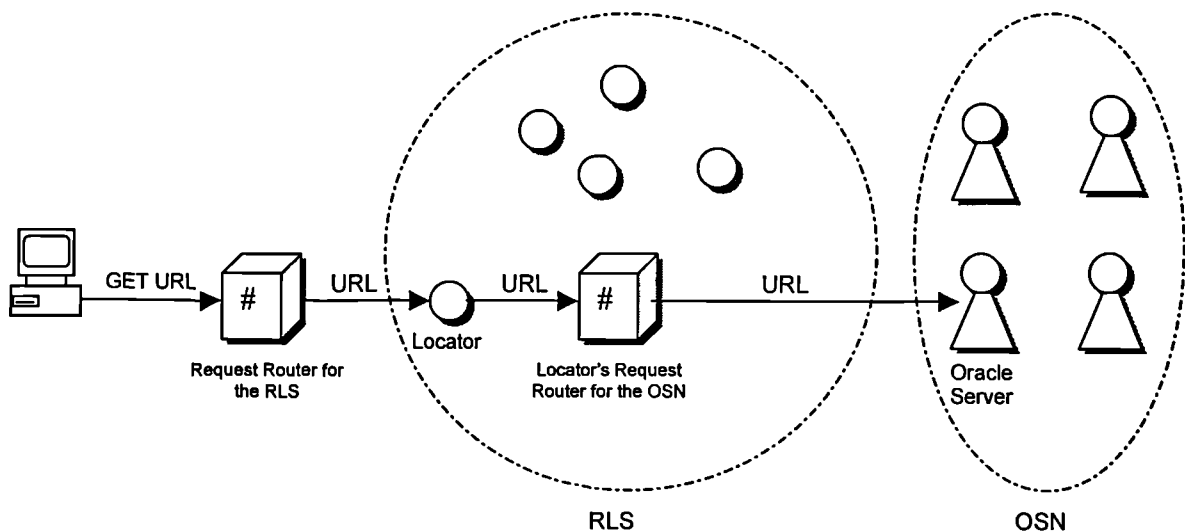


Figure 17 - The Navigational Infons passed from the client to the OSN via the RLS

The infons are guaranteed to be reliable, as there is a direct one-to-one mapping between the user clicking on the hyperlink and the OSN receiving the information from the RLS, with no caching or proxy intermediaries in between that can distort the data. Each Locator in the RLS can locate the appropriate Oracle Server using a RR with the URL pattern set to that of the OSN. In this way, the same URL is used end-to-end from the user to the OSN.

5.5.3.2 Characteristic Infons

Characteristic infons are provided by the resource owner whenever they register their resource with the RLS. The infons should describe the characteristics about the content of the resource, such as subject type, colour, etc., and the physical characteristics of the resource itself, such as file size. The RLS can then pass these details onto the OSN once registration is complete.

The meta-data that should be stored is currently left up to the resource owner, who is free to specify whatever characteristics of his resource that he feels is most appropriate. However, a core set of meta-data will be specified in future research, providing the user with a set of characteristics that should be common for all resources. It is envisaged that this set will be based on the Dublin Core meta-data elements, which include elements such as *Title*, *Creator*, *Publisher*, *Type*, *Language*, etc. (see DublinCoreWG, 1999). However, the user will not be required to submit this information, only encouraged, as it may be a long and laborious process that could discourage resource owners from using the RLS and OSN. In addition, meta-data can easily be forged, and making its submission a requirement of registration would only serve to encourage unreliable information. However, some meta-data can be automatically inferred, such as file-size, content type, etc. Equally, it is in the resource

owner's best interests to submit accurate information if they wish the resource to receive as much attention as possible. The browser will be used by many users to judge the quality of a hyperlink before giving the referenced resource any attention. This alters the selection pressures on the hyperlinks in the user's favour, making those hyperlinks that do have accurate meta-data appear more reliable than those that do not. As such, resources without meta-data will be presented to the user as such (Figure 16 showed them appearing highlighted yellow), and so stand less of a chance of being clicked than those that do. In this way, permitting the user to ignore meta-data acts in the user's interests, rather than against them.

5.5.4 The OSN as a Platform for New Services

The OSN provides a database that contains information about the content on the web and its patterns of usage, which can be retrieved at any time through a standard interface. As such, how this information is presented to the user is dependent upon the client application, and is not restricted to the example in Figure 16. For example, a browser can be configured to allow the user to request the meta-data for each hyperlink, enabling them to see the owner of the underlying resource, for example, rather than just a coloured hyperlink. Alternatively, the user could request the usage statistics for the hyperlink, showing how many people have downloaded the resource over time. In this way, the OSN provides detailed information on the content in the resources, significantly enhancing the user's navigational heuristics.

In addition, the OSN can be seen as a platform for new services from third parties that make use of its information. For example, these services could show how the usage patterns for a resource change over time as its content is updated, or as it migrates across servers. They

could also show how the number of hyperlinks that reference a resource change over time, and compare it with others so that resource owners can see how much attention their resources receive compared with those of their competitors.

The interface to the OSN has not been defined, but will be based on XML messages sent over HTTP. The information that can be queried through this interface has so far only been specific to an individual resource, using its URL as an index into the database. However, future work will look at the viability of enabling each Oracle Server to serve collated information on a range of resources according to complex queries. For example, the OSN, if its information could be co-ordinated, could be used to determine the most popular resource in any given subject; or who the most prolific web resource creator is; or what the biggest resource on the web is, etc. As such, the OSN can be used not just to query the web, but to *measure* it as well, providing a platform for new services that will be able to provide information from across the web as a whole, rather than from a single server. In this way, the OSN increases the web's effectiveness by reducing its noise, and increases its value by enhancing the services it can offer.

5.6 *Summary*

This chapter has described the HOMINID model's architectural design, which enables it to be integrated into the web without breaking the web's existing architecture. Specifically, it has presented the design of the Resource Locator Service, Temporal References, and the Oracle Server Network, and described how they can integrate with the web through the Request Router. This object provides transparent access to any distributed system, enabling the

services of the RLS and the OSN to become part of the web in a way that is completely backwards-compatible with its existing architecture. In this way, the implementation of the HOMINID model provides a solution to the web's flaws that the research set out to solve. Specifically, the model:

- provides a solution to link rot and the shrinking namespace of the web through the RLS;
- introduces the dimension of time to the web with Temporal References,
- decreases the level of noise in the web by increasing the quality of its information and enhancing its navigation mechanisms through the OSN.

The Request Router is the key to the implementation of the model, as it is the novel mediator between the web's architecture and the distributed databases of the RLS and OSN, ensuring that the two systems fully integrate without requiring modifications to the web. Because it is based on the CARP algorithm, which has been adopted and deployed by companies such as Microsoft for use in its Proxy Server product, it is mature and stable enough for enterprise-wide commercial deployment, and the variant defined for the RLS should be just as stable on the web. The following chapter presents a prototype of the Request Router that has been tested and measured to further validate its design.

As was said at the beginning of this chapter, due to time and resource constraints, the research has had to focus on the RLS at the expense of the OSN, the design of which has been described but not specified. As such, the next, and penultimate, chapter will focus exclusively on the RLS. Specifically, the chapter will present a detailed specification of the RLS, including the complete resource migration process; several protocols that have been developed as part of its functional design; and a prototype that has been developed in order to test the design and measure the performance of the concepts described.

6. The Resource Locator Service

This penultimate chapter focuses exclusively on the specification and implementation of the RLS, defining its operation, and presenting a fully working prototype to validate its architectural design. New protocols that have been developed as part of the RLS's design are presented, along with performance measurements of the prototype. In addition, services that use the features of the RLS, such as load balancing and fault tolerance, have been implemented to demonstrate its power and flexibility.

6.1 Introduction

In order to validate the design of the HOMINID model, and to show the effectiveness of the Request Router, the RLS has been fully specified as part of this research, and a prototype built to test its performance. This chapter therefore, focuses entirely on the specification, implementation, and operation of the RLS.

The chapter focuses on how the RLS provides automatic transparent resource migration, and how the number of Locators in the system can be updated. These functions have required the development of two new protocols, which are defined in full. The chapter then presents a detailed description of a prototype RLS that has been developed to validate its design. Measurements taken from the prototype are presented before the chapter concludes with a description of several new services that have been developed that use the RLS to demonstrate its flexibility.

6.1.1 Protocol Development

The functionality of the RLS has required the development of two new protocols:

- The Resource Migration Protocol (RMP);
- The Locator Control Protocol (LCP).

Both protocols are based on HTTP, which enables the Locators to be based on existing web servers rather than a completely new type of server. HTTP is an extensible protocol, and

provides three different ways in which its functionality can be extended (see Whitehead and Wiggins, 1998):

- *URL Munging* – commands to the server are appended onto a URL's QueryString (e.g. *http://www.server.com?commandname=value*)
- *Overloading POST* – HTTP's POST method encapsulates parameters that are sent from the client to the server using user-defined HTTP headers.
- *New HTTP Methods* – HTTP methods are the verbal part of the message that defines what it does (such as GET or POST). New methods can easily be created, however, that extend HTTP's functionality.

Of the three approaches, adding new methods is the one that will be used for the RLS's new protocols. URL munging is perhaps the easiest to implement, but it is an inelegant solution that can easily conflict with other URL munging schemes, and which overloads the semantics of HTTP's GET method (Whitehead and Wiggins, 1998). Using HTTP's POST method to introduce a new protocol is perhaps the most common approach that is used to extend HTTP, but again it overloads the semantics of HTTP's POST method, and it also has the drawback of being a security risk. Most firewalls normally allow HTTP POST messages to pass through unchallenged based on the assumption that the message is HTTP-specific, not part of some undetermined protocol (see Cohen et al. (1998) for an in-depth discussion on the security risks of overloading the POST method). In contrast, new HTTP methods use HTTP's own rules for

extending its method set, and so can reuse its existing headers where necessary (Whitehead and Wiggins, 1998). As such, the new protocols will use new methods and headers that are defined according to the syntax and rules of HTTP version 1.1 (Fielding et al., 1999).

6.2 *Migrating Resources with the Resource Migration Protocol*

The Resource Migration Protocol is based on HTTP so that a resource can automatically migrate across existing web servers without them even being aware of the migration process. In this way, the RMP is completely backwards-compatible with all web servers. However, basing the RMP on HTTP does present problems, as HTTP only provides limited file manipulation support, authentication and querying, all of which are necessary if a resource is to be migrated automatically. However, there is no other web-based standard that can be used across all servers in a consistent, backwards-compatible, and reliable way.

Recently, however, a new set of protocols has been developed by the IETF that support full file manipulation using standard HTTP. The Web Distributed Authoring and Versioning (WebDAV) protocol (Goland et al., 1999) is designed to allow remote authoring of resources by extending the HTTP protocol. Complementary protocols are also being developed to provide authentication and access control, and to allow querying of a web server, all based on HTTP. These protocols are designed for group authoring of web resources, but some of the features they provide can be used in the design of the RMP. This represents a novel application of WebDAV that its designers have not previously identified. As such, this section discusses the suitability of WebDAV for resource migration, and shows how it can be used in the design of the RMP.

6.2.1 Applying the WebDAV Protocols to Resource Migration

WebDAV is designed to “...support efficient, scalable remote editing free of overwriting conflicts” (Slein et al., 1998). It was designed to address the lack of such support in HTTP, so that a defined set of standard functions could ensure interoperability amongst all web servers. For the purposes of the RLS, these functions can be grouped into three broad categories (see Slein et al. (1998) for a complete list):

- Security
- Safe File Transfer
- Server Querying

The following sub-sections describe the WebDAV functions specific to each of these categories, and show how they can be applied to resource migration.

6.2.1.1 Security

WebDAV is designed to enable remote distributed authoring of web resources. As such, it is imperative that only authorized people are allowed to write to a resource. Equally, WebDAV supports file locking on remote web servers, and so requires clients to be authenticated before locks are granted. RFC 2518, the WebDAV RFC (Goland et al., 1999), states that existing HTTP basic authentication (Franks et al., 1999) must not be used, as it is not secure enough, and that HTTP Digest access authentication (Franks et al., 1999) must be used instead. As

such, WebDAV also has a related protocol called the Access Control Protocol (ACP) (Sedlar and Clemm, 2000), which provides greater access control to resources. The combination of HTTP digest and the ACP enable web resources to be safely authored on web servers.

6.2.1.1.1 Application to Resource Migration

The same security concerns that WebDAV has addressed also apply to resource migration. Just as a client must be authorized before being allowed to write to a resource, so a client must be authorized before being allowed to move a resource from one server to another. Moving a resource without permission can be considered theft. Equally, moving a resource onto a server that does not wish to host it can be considered trespass. As such, the client must be authorized before they are granted the right to move the resource, and the destination must grant access before hosting the resource. The ACP can be used to enforce these requirements.

6.2.1.2 Safe File Transfer

HTTP provides rudimentary file transfer features that enable a resource to be downloaded, uploaded or deleted. However, distributed authoring needs more control than HTTP can provide. For example, HTTP does not enable a resource to be moved, only copied (Fielding et al., 1999). Equally, HTTP cannot lock files, and so suffers from the *lost update* problem (Nielsen and LaLiberte, 1999), in which two or more parties updating a resource will inadvertently overwrite previous versions, and thus lose any updates created by the other party. WebDAV was designed specifically to overcome these problems, and does so by extending HTTP to incorporate new methods for file locking, moving and copying.

6.2.1.2.1 *Application to Resource Migration*

A variation of the lost update problem also affects resource migration. For the process to be truly transparent, the resource must be accessible at all times, even when the resource is in the middle of migrating from one server to the next. With HTTP, however, a resource could potentially be updated during the migration process, causing the wrong version to be updated and migrated.

6.2.1.3 *Server Querying*

HTTP provides rudimentary querying of a web server, but only in relation to content negotiation (Fielding et al., 1999). A client can request different versions of the same resource according to the resource's media-type, language, etc., or the client's own capabilities. However, there is no standard interface for querying a web server according to a resource's properties. For example, a client cannot determine a resource's author, or its subject matter, unless it uses a non-standard protocol on top of HTTP that both the client and server support.

WebDAV resolves this by enabling a set of properties to be associated with a resource, using a new HTTP method called *PROPPATCH*, and queried using a new HTTP method called *PROPFIND* (Goland et al., 1999). An associated protocol, DASL (DAV Searching and Locating (Reddy et al., 1999)) provides a new HTTP method, *SEARCH*, for explicitly searching through these properties, enabling them to be fully queried, updated and deleted, all using the ACP to ensure proper authorization.

6.2.1.3.1 Application to Resource Migration

Resource properties are important to resource migration, as they can be used by the destination server to determine whether or not it wants to host the resource. For example, a server may wish to deny a request to host a resource based on the resource's author or subject. In addition, the same properties form the characteristic infons of the Oracle Server (see section 5.5.3.2). Equally, an automatic migration mechanism must automatically update a resolution service. This can be achieved through treating each resource's name/location mapping in a Locator as a resource, with the name and location acting as properties of the resource. Thus, updating a resource's location within the Locator becomes a matter of updating the resource's properties within it.

6.2.2 Disadvantages of Using WebDAV for Resource Migration

The majority of resource migration protocols are used within distributed processing systems, such as RM-ODP implementations (ISO/IEC, 1993). As such, these systems implement the protocol using binary Remote Procedure Calls (RPCs), which are far more efficient and opaque than the text-based messages of protocols based on HTTP. However, these systems generally migrate objects or processes, which require sophisticated handling mechanisms to ensure that the object's code, data, and its current state are all migrated safely. In contrast, web resources are far simpler, largely consisting of static HTML documents or images. Objects and applets can be migrated, but the protocol described here provides no support for migrating an object that is currently executing. Resources are therefore considered as simple files during the migration process. It is, however, feasible for an executing object to be migrated using another protocol in combination with the RMP. For example, Microsoft's

.NET technology (Microsoft, 2001) provides a distributed component-based architecture that enables its components to communicate using an XML-based messaging format known as the Simple Object Access Protocol (Box et al., 2000), which works over HTTP. As such, .NET components could be migrated over the web using a combination of SOAP and the RMP. This functionality, however, will have to wait for future research.

Another disadvantage of WebDAV is that, although it provides MOVE and COPY methods for moving and copying a resource, these have not been defined for cross-server implementation; that is, a MOVE, for example, is only defined for migrating a resource to a new location on the *same* server. This limitation prevents these methods from being used in the RMP, as WebDAV servers will not support moving a resource onto a different WebDAV server. As such, standard HTTP GET, PUT, and DELETE methods will be used to move a resource. WebDAV is used principally to protect a resource during the migration operation (using LOCKs and the ACP), and to update a resource's and Locator's properties.

6.2.3 The Specification of the Resource Migration Protocol

Despite its disadvantages, WebDAV and its associated protocols are ideally suited to automating resource migration on the web. Because it is based on HTTP, it has the benefit of allowing non-WebDAV compliant servers to be involved in the migration process (albeit without the same security and negotiation features available to WebDAV-compliant servers). This section describes the specification of the Resource Migration Protocol (RMP) used for the RLS.

6.2.3.1 *The Migration Process*

The entities involved in the migration process are:

- *The Migration Manager* – oversees the whole migration process, ensuring the availability of the resource at all times, and synchronizing the various stages of the process and the other participants. The migration manager is any entity that wishes to automatically migrate a resource, and can be a client, server, or even the resource itself.
- *The Source* – the original server hosting the resource;
- *The Destination* – the server that the resource will migrate to;
- *The Name Server* – the resolution service entity, whose location properties for the resource must be updated. For the RLS, this is the Locator, but the RMP is generic enough to be used by other name resolution systems that support automatic updates.

A Message Sequence Chart (MSC), showing the sequence of messages for the RMP, is shown in Figure 18, and described in the sub-sections that follow. Note how the Migration Manager is the only participant allowed to act as a client in the whole operation, and how the Source and Destination servers do not communicate with one another at all. The protocol could have been implemented by allowing the Source and Destination servers to communicate directly. A client could send a specially formatted MIGRATE message to the Source, which would then

begin the migration process. However, this would require enhanced functionality in both servers, and so would not be backwards-compatible. Equally, the authentication details of the client would be sent to the Source server, but there is no mechanism in HTTP for the Source to forward them onto the Destination (Fielding, 1996).

The MSC shows a Migration Manager, which is used to manage the migration process. A Migration Manager is any entity overseeing the migration operation, and could be, for example, a dedicated server acting on behalf of the resource owner, a client, or an intelligent agent wishing to migrate itself. The Manager acts on behalf of the resource owner (or resource), and so already has the resource owner's authentication details. Because it acts as a client at all times, it is able to pass on the authentication details to the other entities involved in the migration process, without requiring the details to be forwarded onto another machine. The Migration Manager (also called *the Manager*, for brevity) should be seen as a dumb participant, in that it must explicitly be given the URL of the resource at its Source (URL_{source}), and the new URL, as it would appear on the Destination (URL_{dest}). These details can be passed onto the Manager either through an API call (if the Manager is implemented as an object on the same machine as the entity requesting the migration), or via a new HTTP *MIGRATE* request message, if the entity requesting the migration is on a different machine to the manager. However, the definition of a *MIGRATE* message has been left for future research. The sub-sections that follow describe the operations in more detail.

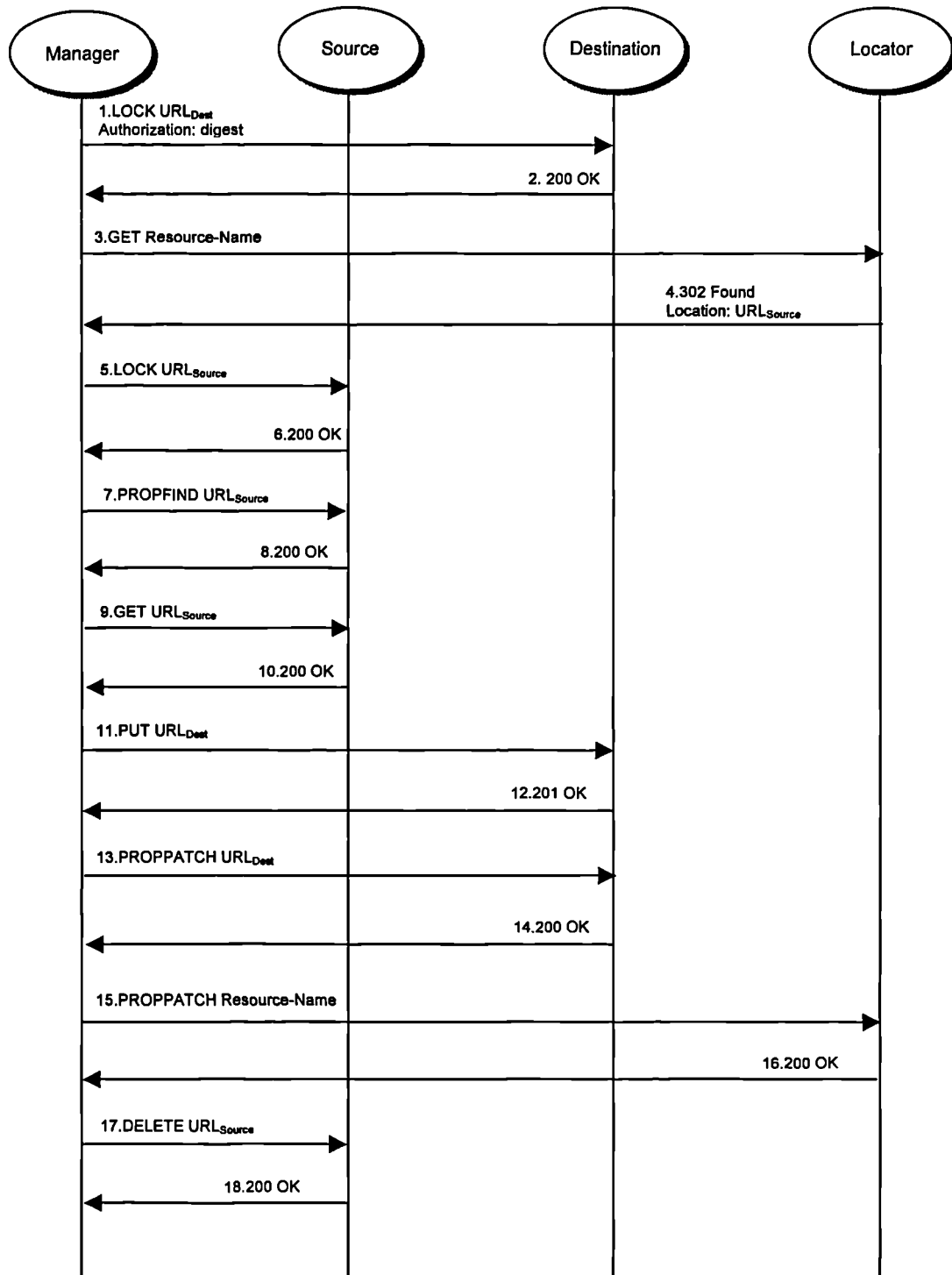


Figure 18 - MSC for Resource Migration Protocol (assuming successful migration)

6.2.3.2 Access Control and Authorization

The migration process begins with the Migration Manager contacting the Destination in order to ascertain whether or not the Destination is willing to host the resource. It does this by sending a WebDAV LOCK message (message 1) to the Destination server for the resource identified by URL_{dest} . As the resource still exists on the Source server, there should be no resource physically located on the Destination that is bound to this location. As such, the Manager is performing a Write lock on a null resource. A null resource is defined as "...a resource which responds with a 404 (Not Found) to any HTTP/1.1 or DAV method except for PUT, MKCOL, OPTIONS and LOCK" (Goland et al., 1999). In other words, a null resource is one that does not physically exist on the server, but whose URL does (link rot can therefore be seen as *null resource creep*). Locking a null resource has the effect of reserving the URL. In this way, a write-locked null resource (or *lock-null resource*) ensures that no other user can use URL_{dest} until the Manager unlocks the resource.

The LOCK message also provides the Destination with the chance to authorize the resource owner. The Manager acts on behalf of the resource owner, and sends authentication details to the Destination server as an HTTP request header in the LOCK message. If the Destination does not authorize the request, it returns a standard HTTP 401 *Unauthorized* response message and the migration process ends. Authorization can also take place prior to the LOCK request using some other authorization scheme accepted by both client and server.

6.2.3.3 Safe File Transfer

The second stage of the process involves moving the resource from the Source to the Destination in such a way that any client can access it at all times. To begin with, the Manager contacts the resource's Locator (message 3) to retrieve the current location of the resource (i.e. to determine URL_{source} - note that Figure 18 uses *Resource-Name* rather than *URL* or *URI*, as the location can be any string). The Locator responds with the location contained within a HTTP 302 Found redirect message (message 4).

Once the current location of the resource has been successfully retrieved, the Manager contacts the Source and LOCKs the resource (messages 5 and 6). Locking the resource ensures that it is not updated in the middle of the migration operation. The Manager must not migrate the resource until it has been successfully locked. The Manager then sends a WebDAV PROPFIND request to retrieve the properties of the resource (messages 7 and 8), before sending a standard HTTP GET message to retrieve the resource itself (messages 9 and 10). The resource is copied to the Destination using a standard HTTP PUT message (messages 11 and 12), using URL_{dest} as the new location for the resource on the Destination. Once this has been accepted, the resource is physically located on the Destination, and ceases to be in the lock-null state (Goland et al., 1999)

The resource is now physically located on both the Source and the Destination. The Manager continues by sending the resource's properties onto the Destination via a WebDAV PROPPATCH request (messages 13 and 14). The properties contain meta-data such as the

resource's subject, its author, copyright information, etc⁷. At this stage, the Destination may decide it cannot host the resource on content grounds; in other words, it will authorize the resource owner and host their resources (using messages 1 and 2), but it cannot authorize the content contained in this specific resource (for example, it may refuse to host resources about a particular subject). Note that a PROPPATCH cannot be performed on a null resource, and so the Destination only gets the chance to determine whether it wishes to host the resource according to its content *after* the resource itself has been copied onto it. Although not optimal, this method has the advantage of complying with WebDAV, ensuring backwards compatibility with existing WebDAV servers.

If the Destination does not wish to host the resource, it must return a HTTP 403 *forbidden* response, and delete the resource. The Manager must then unlock the resource on the Source, and the migration process will have ended once more. Note that the resource is still located on the Source (which, at this stage, has no idea a migration operation is in progress), and the Locator has not been updated, allowing clients to access the resource without problem.

If the Destination does permit the resource to be hosted, the Manager does not need to unlock the resource on the Destination, as “once a PUT...is successfully executed on a lock-null resource the resource ceases to be in the lock-null state” (Goland et al., 1999).

⁷ The complete content negotiation process will be the subject of future research.

6.2.3.4 Updating the Locator

Once the Destination agrees to host the resource, and sends a *201 OK* in response, the Manager must update the appropriate Locator (or other name server). It does this by sending another PROPPATCH message (message 15), with the resource's *Resource-Name* as the identity of the resource (i.e. its name), and *URL_{dest}* as the property to be updated (i.e. its location). Again, authorization must be performed by the Locator.

Recall that two copies of the resource now exist: one on the Source, and one on the Destination. Only after the Manager receives a 200 OK from the Locator (message 16), indicating the new location property has been received and processed, can the Manager send a standard HTTP DELETE request to the Source, instructing it to delete the resource (message 17). Note that both HTTP and WebDAV define a DELETE method, with the WebDAV method providing more control over what is being deleted, and its associated response message (a WebDAV 207 multi-status response (Goland et al., 1999)) providing details about Lock states. As such, the Manager should accept either the HTTP version (200 OK, 202 Accepted, or 204 No Content, (Fielding et al., 1999)) or the WebDAV version of the DELETE response. Once the DELETE response has been received (message 18), the migration process is complete.

6.2.3.5 Resource Replication

The RLS has been designed to cope with replicated resources. Recall that the HOMINID model treats a resource and its content as one atomic unit. As such, the name of a resource is bound to the content that it encodes. With a replicated resource, therefore, although different

physical resources exist with the same name, the content contained within each replica is identical. Thus, the name of the resource should be identical for all replicas, while the location of the resource should be different for each replica. In this way, the name remains consistent, while the name/location combination remains unique for each record in the Locator's database.

To implement this in the RLS requires only a slight modification to the Resource Migration Protocol: After copying a resource onto the Destination, the Manager should send a PROPATCH message to the Locator, informing it to add a new name/location mapping, rather than update an existing one. Once this has been confirmed, the Manager must not delete the existing resource. In this way, the resource has been safely replicated.

In order to support replicated resources, a Locator must ensure that its database can handle multiple locations for the same resource name, and return a HTTP *300 Multiple Choices* message, rather than the default *302 Found* message, when asked for the location of a replicated resource. The *300 Multiple Choices* message is a standard HTTP 1.1 response message that provides a default location for the resource in the location header, and a list of locations in the body (Fielding et al., 1999). In this way, the browser will, by default, simply use the location provided in the *location* header, but other clients are free to use the alternative locations of the replicas according to their own specific requirements.

6.2.3.6 Resource Migration using Non-WebDAV Compliant Servers

The above scenario is specific to resource migration across servers that are WebDAV compliant. If a server is not compliant, however, it can still participate in the migration process, but with less control than with a WebDAV server. For example, WebDAV servers can use the Access Control Protocol to grant or deny access rights, whereas a standard HTTP server must rely on HTTP authentication schemes. Equally, a WebDAV server can read the meta-data associated with a resource, and use this to determine whether it wishes to host the resource or not. A HTTP server, however, cannot, and so must accept or deny the resource based solely on whether or not the resource owner is authorized to upload resources⁸.

Because of the design of WebDAV, a non-compliant server will still be able to communicate with the Migration Manager, even though it does not recognize WebDAV methods such as LOCK. In this case, the Manager will receive a HTTP *405 Method Not Allowed* message (Fielding et al., 1999), and can infer from that that the server does not support WebDAV. As such, it can continue the process using standard HTTP commands only, but needs some way of persisting the resource's properties onto the non-WebDAV server. To do this, the Manager can make use of the fact that these properties are in XML format in the body of a WebDAV PROPFIND message. The body of this message can be saved as an XML file on the Destination, with the same name as the resource, but with *PROP* appended. When the Manager needs to migrate the resource again, the Destination becomes the Source.

⁸ Note that HTTP does contain a framework for content negotiation (see Fielding et al, 1999), which, along with several other such frameworks (including Klyne, 1999; Holtman and Mutz, 1998; and Ohto and Hjelm, 1999), will be examined in future research.

The first time the Manager contacts the Source, it uses a WebDAV LOCK message (see Figure 18). As the Source is not WebDAV compliant, it will return a HTTP 405 *Method Not Allowed* message again, informing the Manager it does not support WebDAV. The Manager should then ensure it GETs not only the resource, but the associated XML property file as well. If the new Destination is WebDAV compliant, the property file can be embedded within the body of a WebDAV PROPPATCH message. In this way, properties can persist across all servers involved in the migration process, regardless of their compliance with WebDAV.

6.3 *Reconfiguring the RLS via the Locator Control Protocol*

As was said in section 5.3.6, traditional hash-routing functions are inherently brittle, and do not adapt well to any changes in the configuration of the system in which they operate. Although robust hash routing algorithms exhibit positive scaling, changes in the system's configuration must still be managed with care, as a Locator that cannot be accessed by a client cuts off access to the resources that it manages.

When a new Locator is introduced into the RLS, it automatically invalidates $1/n$ (where n = total number of Locators) of all name/location mappings managed by the RLS. Because the Request Router automatically updates itself upon the reconfiguration of the system, once it notices the existence of the new Locator, $1/n$ of all subsequent requests to the RLS will go to the wrong Locator, unless the name/location mappings are migrated to the correct Locator without the RR noticing. The automatic update feature of the RR is one of its greatest strengths, but it also means that the RLS must carefully manage transparent *record migration*

(termed to reflect the fact that it is individual records in a Locator's database that must migrate) if it is to remain robust in the face of a changing configuration.

To manage this migration, a new protocol called the Locator Control Protocol (LCP) has been developed as part of this research programme, that allows all Locators in the RLS to be controlled such that records that are located in the wrong Locator can be corrected upon the RLS's configuration changing, without any RR noticing until after the correction has been made.

6.3.1 *The Record Migration Process*

The LCP must ensure that a Locator can be added to or removed from the RLS transparently, such that a RR is able to access all records throughout the system's configuration change. The key to achieving this is to enable both configurations to co-exist for a short period by copying those records that must move to a new Locator *before* the existing configuration is deleted to make way for the new one. In this way, all records are accessible whether the RR chooses to use the RLS in its old configuration or its new one (i.e. with a Locator added or removed from the system).

Resolving the location of a resource given its name is a time-critical process, where latency must be kept to an absolute minimum. In contrast, however, changing the configuration of the RLS is not time-critical at all; as long as all records are fully accessible throughout the configuration change, there is no rush to add or remove a Locator (section 6.3.4 discusses the latency and performance implications of the LCP). The only pressure is to ensure the process

is managed such that the integrity of the system is guaranteed, with all records accessible by all clients throughout the configuration change. The following sub-sections show how the LCP achieves this.

6.3.2 Managing the Addition of a New Locator

6.3.2.1 Overview

When a Locator is added to the system, a RR will only notice the change when it updates itself, and the new Locator has adopted a domain name that complies with the appropriate URL pattern. In this way, the adoption (or removal) of a RLS-compliant domain name acts as a switch: with the domain name, a Locator is recognized by a RR as part of the RLS; without it, the Locator is not recognized, and so will simply be ignored. As such, by first copying all migrating records to their new locations *before* the new Locator adopts its new domain name (Figure 19a), the LCP can enable both configurations to co-exist, ensuring that all records are accessible both before and after the new Locator is recognized by the RR.

The protocol requires the new Locator to act as the record migration manager, coordinating the migration process to ensure integrity of the records. While the migration is occurring, all Locators can still perform their standard name resolution service. Once the new Locator adopts a domain name that complies with the URL pattern, both configurations effectively co-exist. Those RRs that have not updated will be able to access the records in their existing location; those RRs that have updated, will be able to access the records at their new location (Figure 19b). Once in this state, the old configuration can safely be deleted (Figure 19c), causing those RRs that have not updated to receive an *Error 404* when they try to access a

remapped record, which will prompt them to update and thus to recognize the new configuration. In this way, no configuration updates need be sent to any RR throughout the entire process.

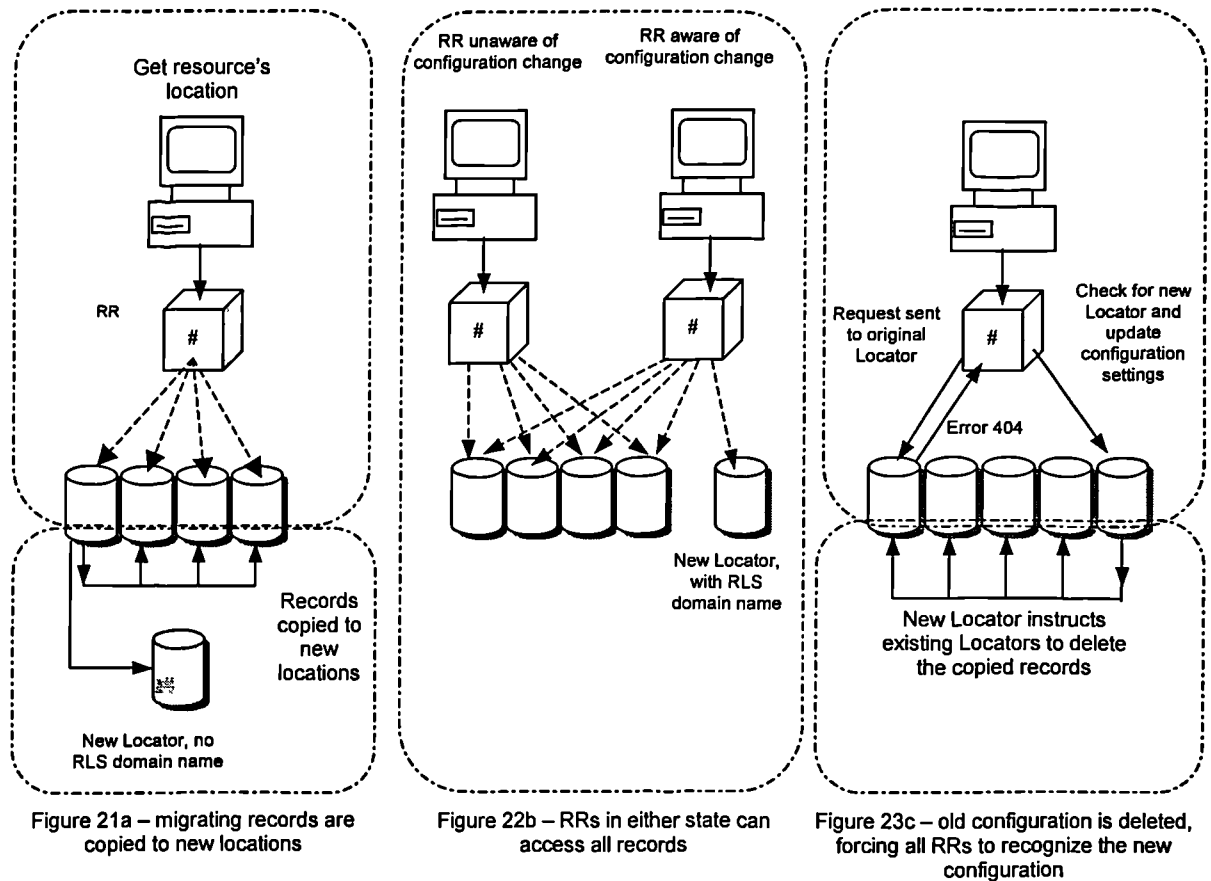


Figure 19a-c - Managing the Addition of a New Locator

6.3.2.2 Message Sequence Chart for Adding a New Locator

For the purposes of the following discussion, the following definitions are made:

- *Node* - A Locator in the RLS.
- *Original configuration* – the number of nodes and the distribution of records in the RLS *prior* to the addition of the new Locator.

- *New configuration* – the number of nodes and the distribution of records in the RLS *after* the addition of the new Locator.
- *Stationary records* – those records in a node that *do not* need to migrate.
- *Mobile records* – those records in a node that *do* need to migrate (i.e. the $1/n$ in each node that must move).
- *The Record Migration Manager* –oversees the whole migration process, ensuring the availability of the resource at all times, and synchronizing the various stages of the process and the other participants. The record migration manager is the new Locator that wishes to join the RLS.
- *Source* – a Locator that correctly stores its records in the original configuration, but which must migrate some of them for the new configuration.
- *Destination* – the Locator that receives the mobile records that the source moves.

Figure 20 shows a Message Sequence Chart that presents the sequence of messages for the Locator addition process managed by the LCP. The MSC is described in the sub-sections that follow.

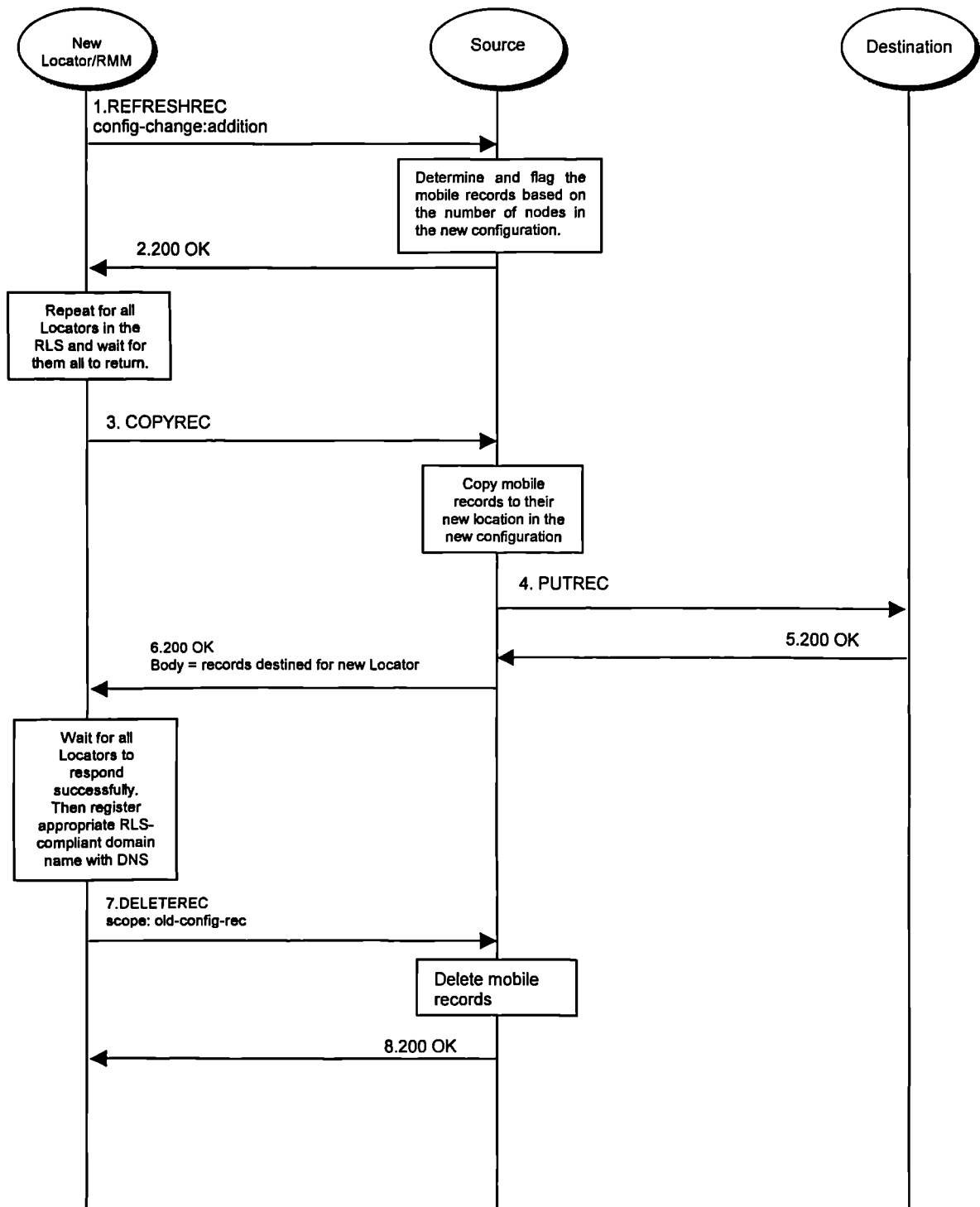


Figure 20 - MSC Describing the Locator Addition Process in the LCP

When adding a new Locator, a RR will only experience problems if it is made aware of the new Locator before the appropriate records have migrated. Equally, a record that is held in the correct Locator as well as in an incorrect Locator will not cause the RR any problems, as it will simply be unaware of the record in the wrong Locator. These facts form the basis for the method that the LCP uses to manage the addition of a Locator to the RLS.

Prior to its addition to the RLS, the new Locator may have a domain name, but this must not conform to the RLS's URL pattern, so as to remain invisible to all RRs. The new Locator (*Locator_{new}*) is defined by the LCP as the record migration manager (RMM), and is responsible for managing the migration of all appropriate records. As such, $1/n$ of all records in each Locator will need migrating, and so *Locator_{new}* will need to know the location of each existing Locator in the RLS. This can easily be achieved by using a standard RR to calculate the number of Locators in the system, and to then calculate their domain names using the RLS's standard URL pattern. Once this has been achieved, the migration process proceeds as follows:

1. *Locator_{new}* must contact each Locator in turn and inform it that a new Locator is about to be added. To do this, it sends each Locator a new HTTP request message with the new method *REFRESHREC* and the new HTTP header *config-change: addition*. *Locator_{new}* is free to send this message to each Locator in turn, or to send it to as many Locators as it wishes in parallel. The only constraint is that all existing Locators in the RLS are sent the message.

The message requires the receiving Locator to update its records based on a changed configuration of the RLS. The *config-change* request header is used to inform the Locator of the addition or removal of a Locator, and to therefore recalculate its records based on the number of Locators that will exist in the new configuration. As such, the receiving Locator (i.e. *Source* in Figure 20) must flag each record in its database that is subsequently found to belong to another Locator in the new configuration, as *mobile*. Note, however, that the Locator must still act as if it exists in the old configuration for all client requests that it receives, and only use the new configuration for determining which of its records must migrate.

2. Upon successful recalculation, Source sends a standard HTTP 200 OK response message back to Locator_{new}, which must wait for all Locators in the RLS to return the same message. If any Locator sends back a response other than 200 OK, Locator_{new} must abort; that is, it cannot continue, and so cannot be added to the RLS until the problem is resolved.
3. Upon successful receipt of all 200 OK messages, Locator_{new} must then send a new HTTP request message with the new *COPYREC* method. This method requires no headers, as it simply instructs the receiving Locator to copy its records that it has marked *mobile* to the appropriate Locator that will host them in the new configuration.
4. Each Source Locator must send its records to the appropriate Destination Locator using the new HTTP *PUTREC* method, with the records encoded as XML in the message's body. *PUTREC* requires the Destination Locator to accept the provided records, and add them to

its database. *PUTREC* differs from HTTP's existing PUT method in that it allows the receiving entity (i.e. Destination) to receive a collection of resources, and to store them as individual records that do not require URIs. In contrast, HTTP's PUT method, as defined in RFC 2616, forbids more than one resource from being encoded in the message's body, and explicitly requires a server to store the single resource under the URI that is supplied in the Request-URI of the PUT message. As such, using PUT at this stage of the LCP would require each individual record to be sent in its own PUT message, and assigned its own URI on the Destination Locator.

The XML format for the body of the PUTREC message is defined as follows:

- *AddRec element* – parent element that signifies the following records are to be added.
- *Rec element* – child element of AddRec that encapsulates one complete record
- *Name element* – child element of Rec that encodes the name of the resource
- *Location element* – child element of Rec that encodes the location of the resource
- *TimeCreated* – child element of Rec that encodes the time that the resource was created, defined according to RFC 1123 (Braden, 1989).

An example PUTREC message is shown in Figure 21.

```
PUTREC HTTP/1.1
Host: Source.Locator.net
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
<?xml version="1.0" encoding="utf-8" ?>
<L:AddRec xmlns:L="LCP:"
<L:Rec>
<L:Name>www.mobilerecord1.com/name/img.gif</L:Name>
<L:Location>www.resourcehost.com/location/img.gif</L:Location >
<L:TimeCreated>Sun, 06 Nov 1994 </L:TimeCreated>
</L:Rec>
<L:Rec>
<L:Name>www.mobilerecord2.com/name/doc.htm</L:Name>
<L:Location>www.resourcehost2.com/location/doc.htm</L:Location >
<L:TimeCreated>Sun, 26 Oct 2000 </L:TimeCreated>
</L:Rec>
</L:AddRec>
```

Figure 21 - Example PUTREC message

5. Upon successfully receiving and storing the records, the Destination returns a standard HTTP 200 OK message.
6. The Source then sends a 200 OK message back to *Locator_{new}*, but with the records that *Locator_{new}* must store encoded in the body of the response using the same XML format as defined for the PUTREC message in 4. Upon receiving this, *Locator_{new}* must store the records in its own database.

Once each Locator has returned a 200 OK message, the RLS is placed in a juxtaposition of two configurations: the original configuration that operates without *Locator_{new}*, and a

new configuration that operates with `Locatornew`. Because the records have been copied rather than moved, both states are active simultaneously. As such, it is safe for `Locatornew` to register a name for itself with the DNS that *does* conform to the RLS's URL pattern, and so make itself visible to all RRs (both configurations are equally valid, and so it does not matter which state the RR perceives). It is envisaged that `Locatornew` uses the new dynamic DNS protocol (Vixie et al., 1997) to achieve this, ensuring the process continues automatically.

7. `Locatornew` sends a DELETEREC message to each Locator with a *scope* header that has the value *old-config-rec*. This informs each Locator to delete all the mobile records that have been copied to a new Locator now that the RLS is in its new configuration. Those RRs that have not updated will receive an Error 404 when they request one of these records, but this will simply force the RR to update. Once it does this, it will be aware of the new system configuration, and will be able to access the records at their new position.
8. Each Locator responds with a 200 OK message. Once all 200 OK messages have been received, the RLS is safely in the new configuration with the new Locator added, and the LCP has ensured that all records have been fully accessible at all times.

6.3.3 *Managing the Removal of an Existing Locator*

6.3.3.1 *Overview*

Removing a Locator requires a different approach as deleting its RLS-compliant domain name may leave a hole in the sequential numbering of the URL sequence, confusing the RRs. The process begins when the detaching Locator (*Locator_{detach}* – coloured black in Figure 22), acting as a RMM, instructs all other Locators in the system that the configuration is about to change, thus causing them to copy the records that must migrate to their new locations. Note that some of the records will be copied onto *Locator_{detach}*, even though it is about to leave the system (Figure 22a). Once this is complete, *Locator_{detach}* remains in the RLS, and instructs the *last* Locator in the sequence (*Locator_{last}* – coloured grey in Figure 22) to detach itself from the system, even though *Locator_{last}* is not the one that wishes to leave (Figure 22b). In this way, the RLS shifts to the new configuration, with the existing configuration still operational (note that the RRs that have not updated may attempt to reach *Locator_{last}*, but will not receive a response; this will cause them to update automatically, however, thus moving them to the new state).

Once the new configuration has been reached, *Locator_{detach}* will instruct the (now removed) *Locator_{last}* to delete all of its records, before copying its own records over to make both Locators mirrors of one another. In addition, *Locator_{last}* will also be given the same domain name as *Locator_{detach}*, making the two Locators identical clones (Figure 22c). Once this happens, *Locator_{detach}* is free to detach itself by removing its IP address from the DNS entry for its domain name, leaving the RLS in the new configuration. Again, the process of

removing a Locator requires no synchronization messages from any Locator, as all RRs will automatically update themselves.

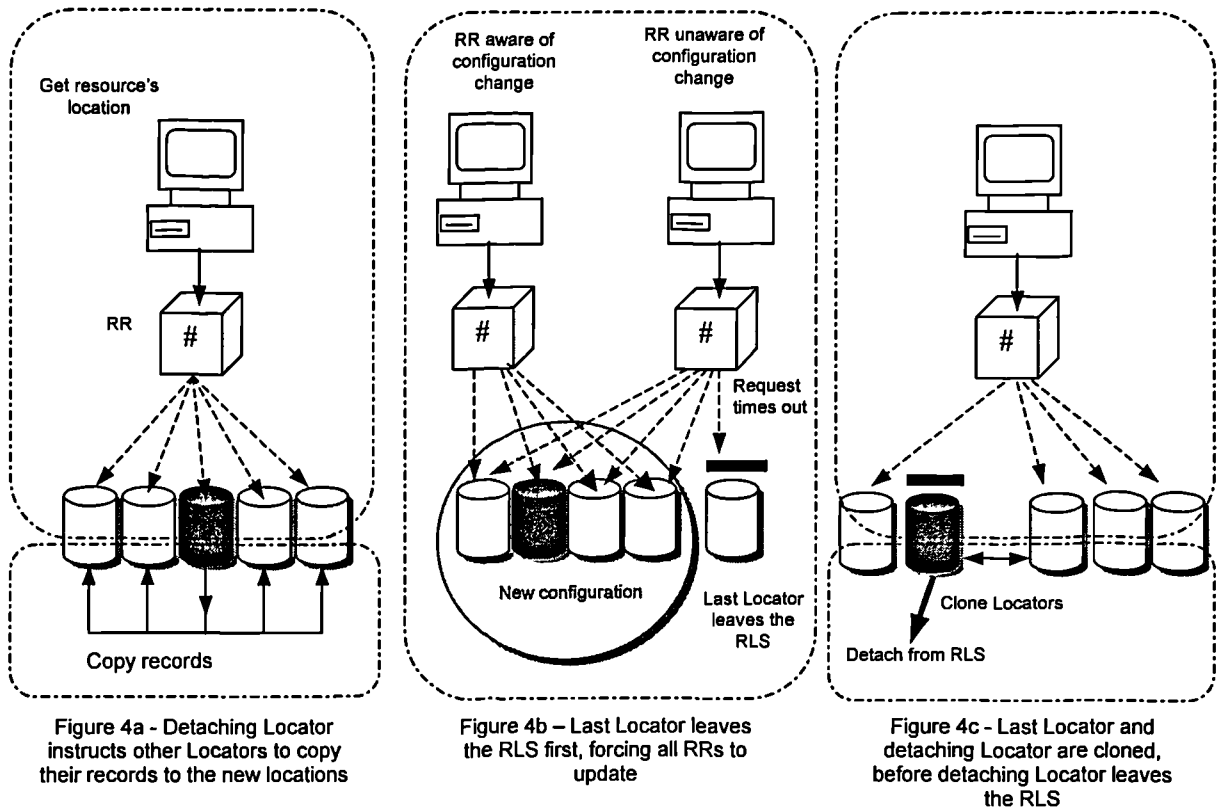


Figure 22a-c - Managing the Removal of an Existing Locator with the LCP

The following discussion and the MSC in Figure 23, describe the removal process in more detail, and define the messages used by the LCP that implement it. Note that the definitions given for section 6.3.2 are still valid.

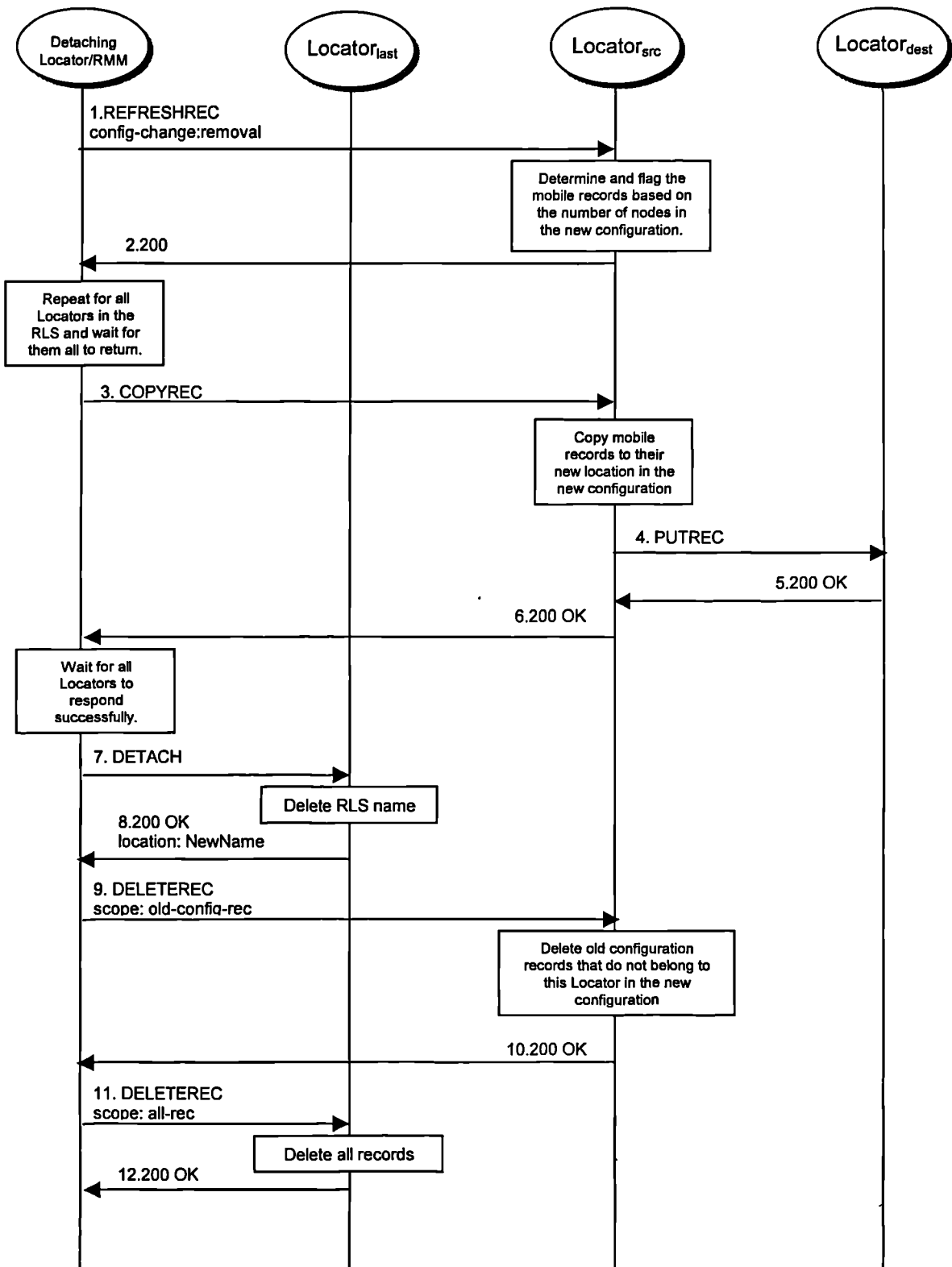


Figure 23 - MSC for Removing a Locator (continued on following page)

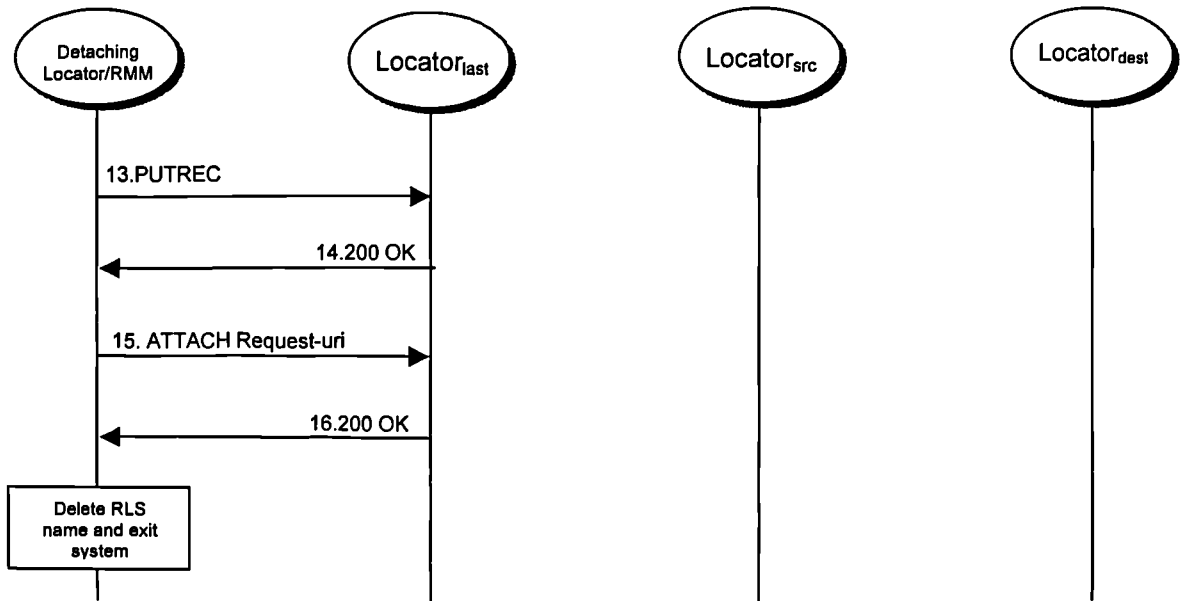


Figure 23 - MSC for Removing a Locator (cont.)

6.3.3.2 Message Sequence Chart for Removing an Existing Locator

The Locator that is about to leave the RLS, *Locator_{detach}* is defined by the LCP as the Record Migration Manager (RMM). As the RMM, it must coordinate the process of removal, ensuring that all records in all Locators are accessible by clients throughout the entire process. It must begin by calculating the domain names of the other Locators in the system by using its RR, and then determine the last node in the system, *Locator_{last}*. Once this is done, the removal process proceeds as follows:

1. *Locator_{detach}* sends a REFRESHREC message to each Locator in the system, but this time the *config-change* header is set to *removal*. Each Locator, upon receiving this message, must decrease the number of nodes that it thinks the RLS has by one, and

recalculate the correct location for its records using the RR, based on this new number. Again, the Locators must assume that the original configuration still persists for any client requests that they receive at this stage.

2. Upon success, each Locator will return a standard HTTP 200 OK message. $Locator_{detach}$ must wait for all Locators to return 200 OK before continuing.
3. $Locator_{detach}$ sends a COPYREC message to each Locator, instructing them to copy their mobile records to the Locator that will host them in the new configuration ($Locator_{dest}$). Note that all records in $Locator_{last}$ will need to be copied, as this will not exist in the new configuration.
4. Each $Locator_{src}$ sends a PUTREC message to the appropriate $Locator_{dest}$ with the contents of its mobile records held in the body of the message (see Figure 21). Upon receipt, $Locator_{dest}$ must store the records in its database.
5. $Locator_{dest}$ sends a HTTP 200 OK confirmation response back to $Locator_{src}$.
6. $Locator_{src}$ sends a HTTP 200 OK confirmation response back to $Locator_{detach}$, which must wait for all Locators to send back the same message before it can proceed. If one Locator does not return a 200 OK, $Locator_{detach}$ must send a new HTTP ABORT message; every Locator receiving this must place itself back in the original configuration, by deleting all newly-copied records from its database.

7. Upon receipt of all confirmation messages, `Locatordetach` issues a new DETACH message to `Locatorlast`, ordering it to detach itself from the system. `Locatorlast` does this by removing its RLS-compliant name from the DNS. Upon doing this, the RLS moves to the new configuration. Those RRs that are not aware of the change will send their requests to the wrong Locator, and will receive an *Error 404*. This will cause them to automatically update to determine the correct number of nodes, whereupon they will discover that the RLS is in a new configuration, with one less node than they had assumed. The RRs will then update the number of nodes they believe the RLS to have, and all new requests will be routed to the correct Locator. Thus, the RLS safely moves over to the new configuration without preventing clients from accessing any records.

8. `Locatorlast` responds to the DETACH request with a HTTP 200 OK message that contains a *location* header, informing `Locatordetach` of the name or IP address that `Locatordetach` should use in future communication with `Locatorlast` (this is required as the RLS-compliant name of `Locatorlast` will no longer be valid, and `Locatordetach` will not have any other name with which to identify `Locatorlast`). Future versions of the protocol should include a *defer-until* header, which gives `Locatorlast` time to delete its RLS-compliant domain name. The header should specify the time by which `Locatorlast` will have removed its RLS-compliant name. However, this disconnected feature of the protocol has not been fully defined at this stage of the research, and will be left for a future version (see section 7.3).

9. Locator_{detach} sends a DELETEREC message to all other Locators apart from Locator_{last} with a *scope* header that has the value *old-config-rec*. This informs each Locator to delete all the mobile records that have been copied to a new Locator now that the RLS is in its new configuration. Locator_{detach} must also delete its own mobile records that have been copied to other Locators.
10. Each Locator responds with a 200 OK message. Once all Locators have returned 200 OK, the RLS is in the new configuration, with all records in their correct Locators, and no duplicates in existence. However, it was Locator_{detach} that wished to leave the system, and not Locator_{last}, and so the final messages are required in order for Locator_{detach} to leave the RLS, and Locator_{last} to reattach itself.
11. Locator_{detach} sends a DELETEREC message to Locator_{last}, but with a *scope* header that has a value of *all-rec*. This instructs Locator_{last} to delete all records within its database.
12. Locator_{last} responds with a 200 OK message.
13. Locator_{detach} then sends one or more PUTREC messages to Locator_{last} containing all of the records that Locator_{detach} currently manages.
14. Locator_{last} stores these records and returns a 200 OK message. Locator_{detach} and Locator_{last} are now mirrors of one another.

15. Locator_{detach} sends an ATTACH message to Locator_{last} with a Request-URI value of the RLS-compliant domain name of Locator_{detach}. Upon receipt, Locator_{last} must register this name as an alias to itself. As soon as it does so, Locator_{last} is part of the RLS once more, but not as the last node in the system. The domain name that Locator_{detach} uses now identifies both Locator_{detach} and Locator_{last}, both of which are mirrors of one another.

16. Locator_{last} sends a 200 OK message back to Locator_{detach}. Upon receipt of this message, Locator_{detach} must remove its RLS-compliant domain name from the system. Once it has done so, it has officially left the RLS, leaving the RLS in the new configuration, with the appropriate Locators still members of the system.

6.3.4 Performance Implications of the LCP

Changing the configuration of the RLS is not a time-critical process, as the name resolution service provided by the RLS is unaffected throughout the change. However, the change should still occur in a reasonable time-frame, and with a reasonable amount of network traffic, and so this section presents an estimate of the order of time that will be needed for a new Locator to be added the system. Note that no estimations are provided for removing a Locator, as the operations are very similar, and so the time taken will be of a similar order.

The addition of a new Locator involves two steps that could significantly affect the time taken to update the system:

- Determining which records need to move;
- Physically moving the records.

The other steps involve data manipulation, such as deleting records, which will not negatively affect the scalability of the design or the time taken to change the configuration, and so will not be considered in the following calculations.

The first step involves every record in the system being processed by a RR whose node configuration is set at one node higher than its current value (i.e. set to $n + 1$). The time taken to do this can be significantly reduced if each Locator works in parallel with its peers, processing only the records contained in its own database. This is how the LCP operates. As such, ignoring the network overhead of the LCP, the time taken for step one will be approximately:

$$\frac{Rt\#}{n+1}$$

where R is the total number of records in the system, and $t\#$ is the time taken to process one record. Thus, for the same R , the time will decrease with the number of Locators in the

system. Section 6.4.5.4 provides an example of the time taken based on figures obtained from the prototype RLS.

The time taken to complete the second step, however, is dependent upon the number of Locators in the system, and the number of records. When a new Locator is added to an n -node system, the records that are re-mapped will be evenly distributed across all Locators in the RLS (Thaler and Ravishankar, 1998). As such, each Locator will evenly distribute $1/(n+1)$ of its own records (which represent $1/n$ of the total number of records in the system) to the n other Locators in the RLS (including the new one). This results in n Locators broadcasting to n Locators (including the newly added one), resulting in the propagation of n^2 messages. As such, the number of messages that this step generates increases with the square of the number of Locators, which could constrain the size of the system. However, the total number of messages, m , is limited by the number of records that are invalidated, as clearly there cannot be more record-carrying messages than there are records to move. Thus:

$$m \leq \frac{R}{n+1}$$

Once m reaches this limit, the number of messages will *decrease* as more Locators are added, improving the scalability of the design markedly.

As such, the number of messages that are broadcast can be controlled by balancing the number of records in the system with the number of Locators. Furthermore, as section 6.4.5.4

demonstrates, the time taken to add a new Locator is bounded by Rp , where p = the time taken to hash one record with only one Locator in the system. That is, once m has reached its limit, the time taken to add a new Locator becomes independent of the number of Locators in the system. This is an important result, as it proves that the scalability of the system depends only on the processing power of the machines performing the hash routing algorithm, and not on the configuration of the system itself. Furthermore, the LCP can also be optimised so that it broadcasts messages in parallel across Locators according to bandwidth available and Locator performance, cutting the time it takes to broadcast the messages. In addition, each record will only be of the order of 150 bytes (assuming an average of 50 characters each for the name, location, and time of creation values), resulting in a relatively small amount of data that must be transferred regardless of the number of messages. Finally, it is worth reiterating that a configuration change is not a time-dependent task, as the RLS is fully operational throughout the change, and it will not happen often, as the configuration of the RLS should remain stable for relatively long periods of time. *As such, the design of the RLS is such that network overhead never becomes a limiting factor, whatever the number of Locators in the system.* The proof of this is provided in section 6.4.5.4, which presents sample calculations illustrating the order of time that is required to add a new node to a system using a variety of different configurations.

6.4 *A Prototype Resource Locator Service*

In order to validate the design of the RLS and the RR, a prototype has been built and tested as part of this research.

The prototype Resource Locator Service comprises:

- a small network of Locators;
- a Request Router;
- a HTTP proxy server;
- a management interface.

Figure 24 shows the architectural design of the prototype RLS. The design differs slightly from the architecture presented in section 5.3.4 (see Figure 14), as the proxy server retrieves the resource from the origin server, rather than the client. However, the RLS's architecture does not specify where the RR is hosted, or the semantics of the RR's host, as these will change according to where the RR is hosted. As such, using the proxy server to retrieve the resource simplifies the implementation considerably, and because the proxy is not placed under a heavy load, the performance of the system remains unaffected.⁹

The following sub-sections provide more details about the implementation of each component in the prototype, while section 6.4.5 presents performance measurements that were taken to validate its design.

⁹ Note that the performance would be affected if many clients used the proxy, as it is not a scalable design.

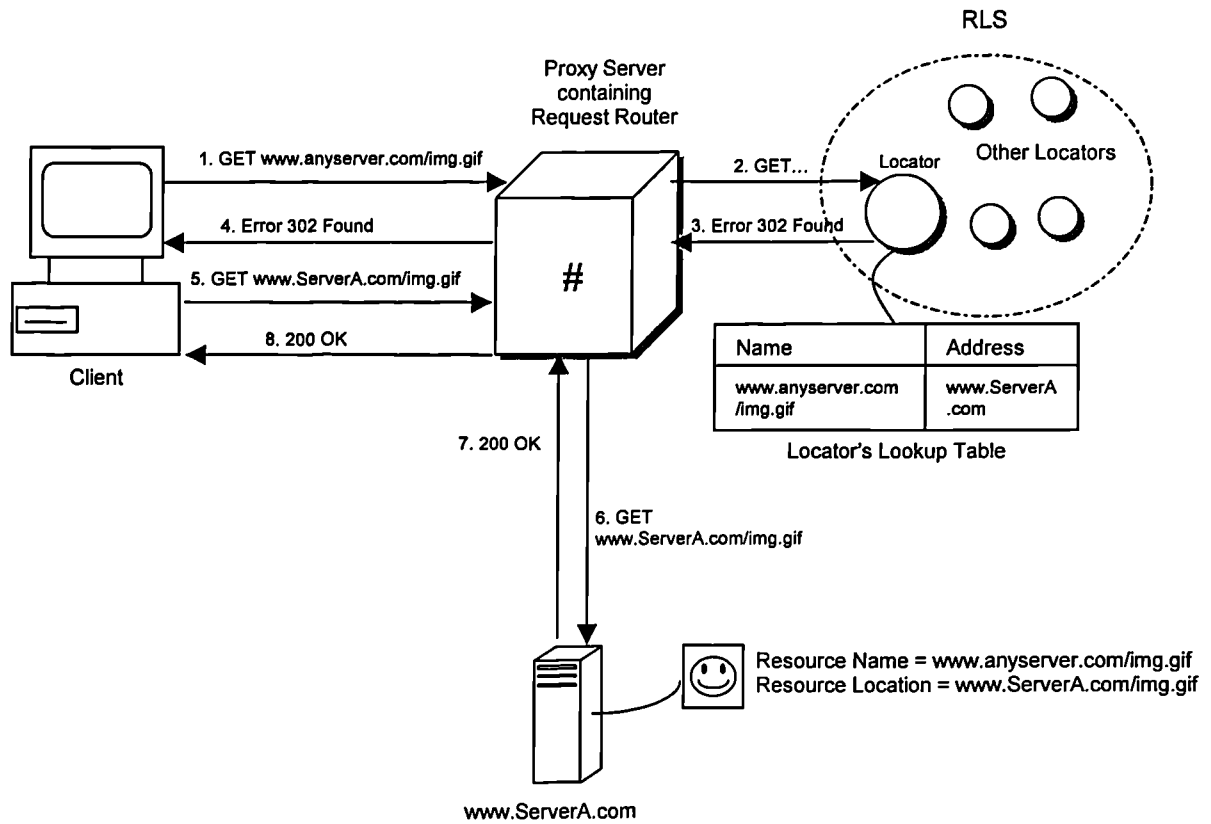


Figure 24 - Architectural Design of the Prototype RLS

6.4.1 A Prototype Locator

The Locator has been designed as a web server using Microsoft IIS on NT 4 Server, which uses Active Server Pages (ASP) to implement a Locator's functionality through integration with a Microsoft Access database via ODBC. The database stores the following for each resource:

- Resource's name (this can be any string).
- Resource's *current* location (this must be a URL).

- Resource's time of creation.
- Sequence Order (used for load balancing purposes; note that this is not part of the functional design of the RLS, but has been included in the prototype to demonstrate some applications of the RLS).

The same resource name can reference multiple entries in the database, as each resource may have multiple locations (i.e. replicated resources) and multiple times of creation (i.e. when a resource's content is changed but its name remains unchanged).

When a Locator receives a standard HTTP request from a client, it looks up the resource in its database. If it contains the resource's name/location mapping, it returns a HTTP *302 Found* response message; otherwise it returns a HTTP *Error 404 Not Found* error message. In this way, a client can communicate with the RLS transparently, providing full backwards compatibility.

If the URL of the requested resource was encoded as a Query String, or if the new HTTP request header *authoritative-lookup* is used (see section 5.3.5.1), the Locator performs an authoritative lookup, using the RR itself if it cannot locate the resource in its own database to ensure the client has reached the correct Locator. An authoritative lookup is guaranteed to locate the resource's name/location mapping, provided the resource is registered with the RLS.

If the Locator receives a HTTP request with a HEAD method, it will simply return a HTTP *200 OK* response. This is used so that RRs can safely query for the existence of a Locator.

The database can be queried using standard WebDAV PROPFIND messages¹⁰. The PROPFIND method allows a resource to be queried according to its attributes. In this way, each name/location mapping (and corresponding information) in the Locator is treated as a resource, enabling the location of managed web resources to be returned according to their name, location or time of creation.

Finally, the database can be updated automatically and remotely using the Resource Migration Protocol and standard WebDAV and HTTP commands. Due to time and resource limitations, temporal references have not been included in the prototype, but their implementation will not be difficult, and will be left as future work.

6.4.2 A Prototype Request Router

The Request Router is perhaps the most important part of the system, as it has to integrate into the web's existing architecture. To do this, a Request Router object has been created in C++ and embedded into a simple HTTP proxy server. Any user who wishes to use the RLS can configure their browser to use the proxy server, enabling all legacy browsers and servers to use the RLS transparently.

¹⁰ The prototype Locator has been designed to support only a minimal subset of the WebDAV PROPFIND semantics, and so although it supports PROPFIND, it does not implement it according to the WebDAV specification.

The Request Router object can be deployed on any system where a developer has source code access. However, it is trivial to turn the object into an ActiveX control, in which case it can be embedded into any Microsoft Windows application that supports ActiveX controls (for example, Internet Explorer, the Microsoft Office suite of products, or any application that supports scripting). For more cross-platform support, the component could have been written in Java and turned into a JavaBean, enabling it to be deployed on any platform. However, for the purposes of the prototype, it need only exist on a Microsoft Windows NT platform, and so Microsoft Visual C++ was chosen as its development language.

6.4.3 A Prototype Management Interface

A management interface has been developed for the prototype system (Figure 25) that provides a user interface for managing resources on web servers, and for demonstrating certain features of the RLS and the RMP. The interface acts as the Migration Manager, implementing the RMP and co-ordinating messages amongst the web servers being managed. It will work with any web server, whether it is WebDAV compliant or not, so long as each grants write access to the Manager (username and password settings can be stored within the management interface). Two of the servers in the prototype use WebDAV, and two do not. None of them are aware of the resource migration protocol, or that they are involved in a migration operation, demonstrating the backwards-compatibility of the service.

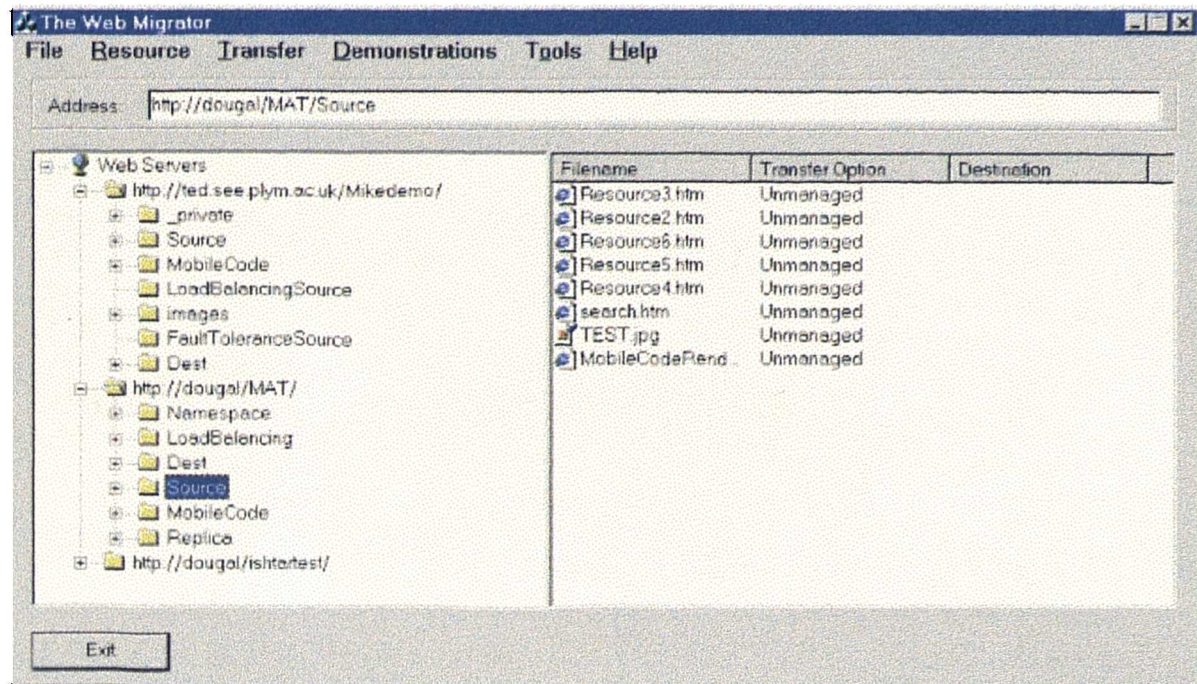


Figure 25 - RLS Management Interface

6.4.4 Implementing the Resource Migration Protocol

The implementation of the protocol uses Microsoft's OLE DB Provider for Internet Publishing (Microsoft, 2000b). This is an API that allows clients to manipulate resources and properties on remote servers using WebDAV messages. As such, the API handles all WebDAV message processing, allowing the development work to focus on the implementation of the Resource Migration Protocol.

The Management Interface provides a Windows-explorer like user interface that shows the directories of the managed web servers as if they were part of the local file system. Resource migration is handled through a drag and drop interface, which enables the user to drag a web resource from its source directory on one web server, and drop it onto a destination directory on another server, in the same way that files are copied and moved using Windows Explorer.

The migration process is managed by the Management Interface using RMP, which ensures that the correct Locator is updated automatically, such that any client using the proxy server is able to access the resource at all times throughout the migration operation.

6.4.5 Performance

The prototype RLS was designed to functionally validate the design of the RLS, and it has achieved this by showing how standard web resources can be transparently migrated across any existing web server, without the modification of any client. In addition, the prototype has also been instrumented to provide performance measurements of the RLS. The results of these measurements are presented in the following sub-sections.

6.4.5.1 Network Overhead

Network overhead will always be two extra HTTP messages (one request and one response), regardless of the size of the system. As such, network overhead on its own has not been measured, as it does not impact on the scalability of the system.

6.4.5.2 CPU Overhead

The Request Router was tested on a Pentium Pro 200MHz with 64MB RAM, a Pentium III 400MHz with 128MB RAM, and an Athlon 1100MHz with 128MB RAM. The RR was designed so that the number of nodes it believed existed within the RLS could be manually set, and instrumented to enable it to measure the length of time it took to identify the correct Locator. The results are presented in Figure 26, which clearly reveals the linear relationship between time and the number of Locators. The results show that for small numbers of Locators, the time taken is insignificant, and that even with more Locators, the time taken is

still small. As such, even with a relatively slow machine such as the Pentium Pro 200MHz, the RR can determine the correct Locator from a 10,000-node configuration in only 0.35 seconds.

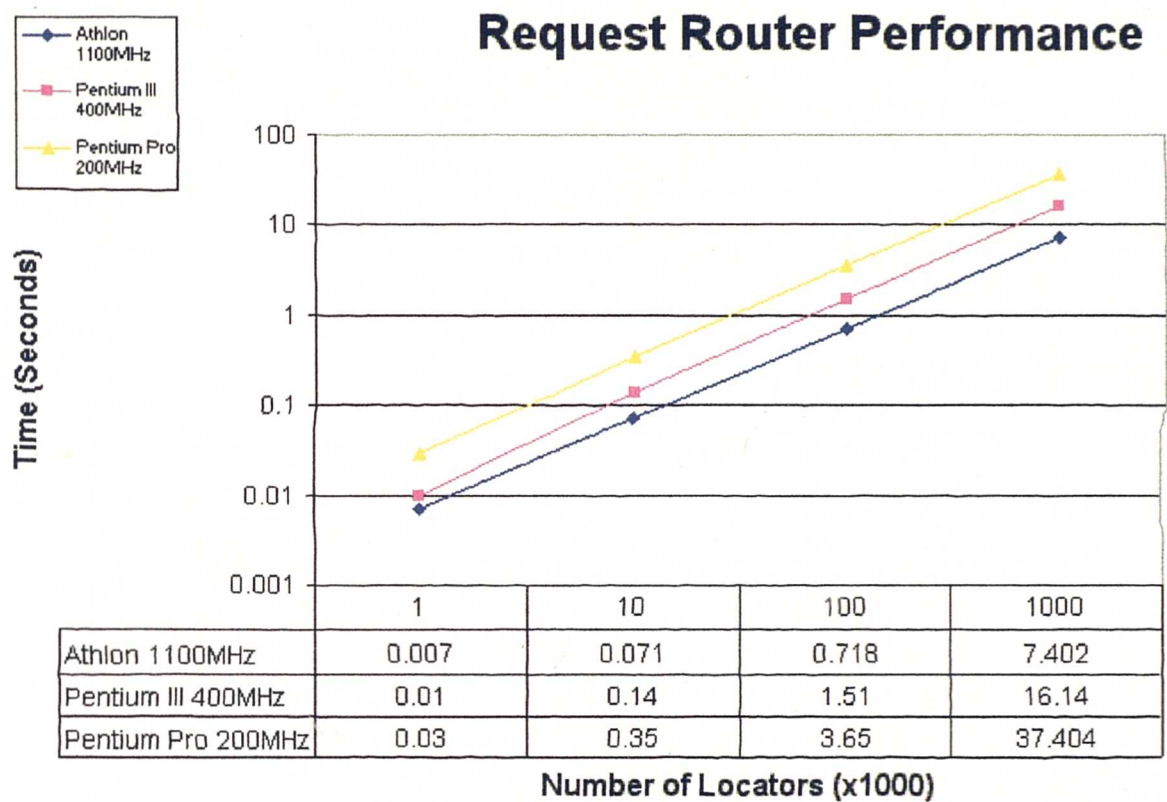


Figure 26 - Performance Results of the Prototype Request Router

In addition, the prototype RR was designed for experimental purposes, and is non-optimal. Specifically, it rehashes every Locator URL for every request that it routes, but unless the number of Locators changes, these hash values will remain static. As such, a more optimal

design would cache the hash values in memory, and only rehash them when the configuration changes, thereby drastically reducing the length of time it would take to locate a Locator.

6.4.5.3 Total System Overhead

The total overhead introduced by the system was measured to provide a real-world indication of the system's performance. To do this, the time taken to visit the homepage of three different web sites (*www.google.com*, *www.lycos.co.uk*, and *www.yahoo.com*) was measured using a standard browser and no RLS. Each site was visited 25 times, with the browser's cache deleted each time. The browser was then connected to the RLS via the proxy server, and the same sites were visited 25 times again. The experiment was run using an Athlon 1100MHz PC with 128MB RAM, which acted as the proxy server with an embedded RR, and a Pentium Pro 200MHz PC with 64MB RAM, which encoded the functionality of the Locator. Both machines used Microsoft Windows NT 4 Server, and were connected via a 10Mbps Ethernet LAN.

For this experiment, the number of Locators was varied in the RR from one to 1 million. However, to avoid having to physically deploy 1 million Locators, the proxy server was configured so that it always forwarded the request onto the same Locator, regardless of which one the RR identified. The Locator would then return an *Error 404* message, which would cause the proxy server to redirect the request to the origin server, from where the resource would ultimately be retrieved. Because the overhead for the RLS is the same whether the resource is found or not (i.e. one extra HTTP request and one extra HTTP response), the measurements of the overall system overhead remained unaffected. In addition, this

configuration removed any differences between servers that would have been introduced had each Locator been deployed on a separate physical machine.

The results presented in Table 6 show the time taken to visit each web site without the RLS, and the time taken to visit it with the RLS, with one, 1,000, 10,000, 100,000, and 1,000,000 Locators in the system. Each value is the 10% trimmed mean of 25 trials, with the overhead calculated by subtracting the mean from the value obtained without the RLS. The results show the overhead introduced by the RLS varies from 0.7 to 0.87 seconds with only one Locator in the system; from 1.42 to 1.58 seconds with 100,000 Locators; and from 8.10 to 8.33 seconds with one million Locators in the system. The results are consistent across the different web sites, with the overhead introduced by the RR only becoming noticeable with 100,000 Locators, and becoming impractical on 1,000,000 Locators.

	<i>www.google.com</i> (Standard download time = 3.660 seconds)		<i>www.lycos.co.uk</i> (Standard download time = 7.610 seconds)		<i>www.yahoo.com</i> (Standard download time = 6.192 seconds)	
Number of Locators	Download time (seconds)	Overhead (seconds)	Download time (seconds)	Overhead (seconds)	Download time (seconds)	Overhead (seconds)
1	4.363	0.703	8.477	0.867	7.063	0.871
1000	4.370	0.710	8.483	0.873	7.070	0.878
10,000	4.434	0.774	8.546	0.936	7.135	0.943
100,000	5.081	1.421	9.190	1.580	7.775	1.583
1,000,000	11.765	8.105	15.985	8.375	14.524	8.332

Table 6 - Results of the Overhead Introduced by the RLS

The results show that the RLS introduces negligible overhead for a configuration of 10,000 Locators or less. However, it should be noted that neither the design of the RR or the proxy

server are optimal, and that significant performance improvements can easily be made. Such improvements are expected to enable the deployment of a 100,000 Locator system with negligible overhead.

6.4.5.4 The Cost of Changing the Configuration

As section 6.3.4 discussed, changing the configuration of the RLS incurs a performance cost. Using the figures above, this cost can now be calculated. The following scenario represents a new Locator being added to a RLS designed for today's web. As such, the total number of resources managed, R , is 1 billion (10^9), and the number of Locators, n , in the *original* configuration is 999. Recall that changing the configuration comprises two steps:

- Determining which records need to move;
- Physically moving the records.

The first step takes $\frac{Rt\#}{n+1}$ seconds. From Figure 26, $t\#$ takes 0.007 seconds for an Athlon

1100MHz machine, which would lead to a total time of:

$$\frac{10^9 \times 0.007}{1000} = 7000 \text{ seconds (or 1 hour 57 minutes, 7 seconds).}$$

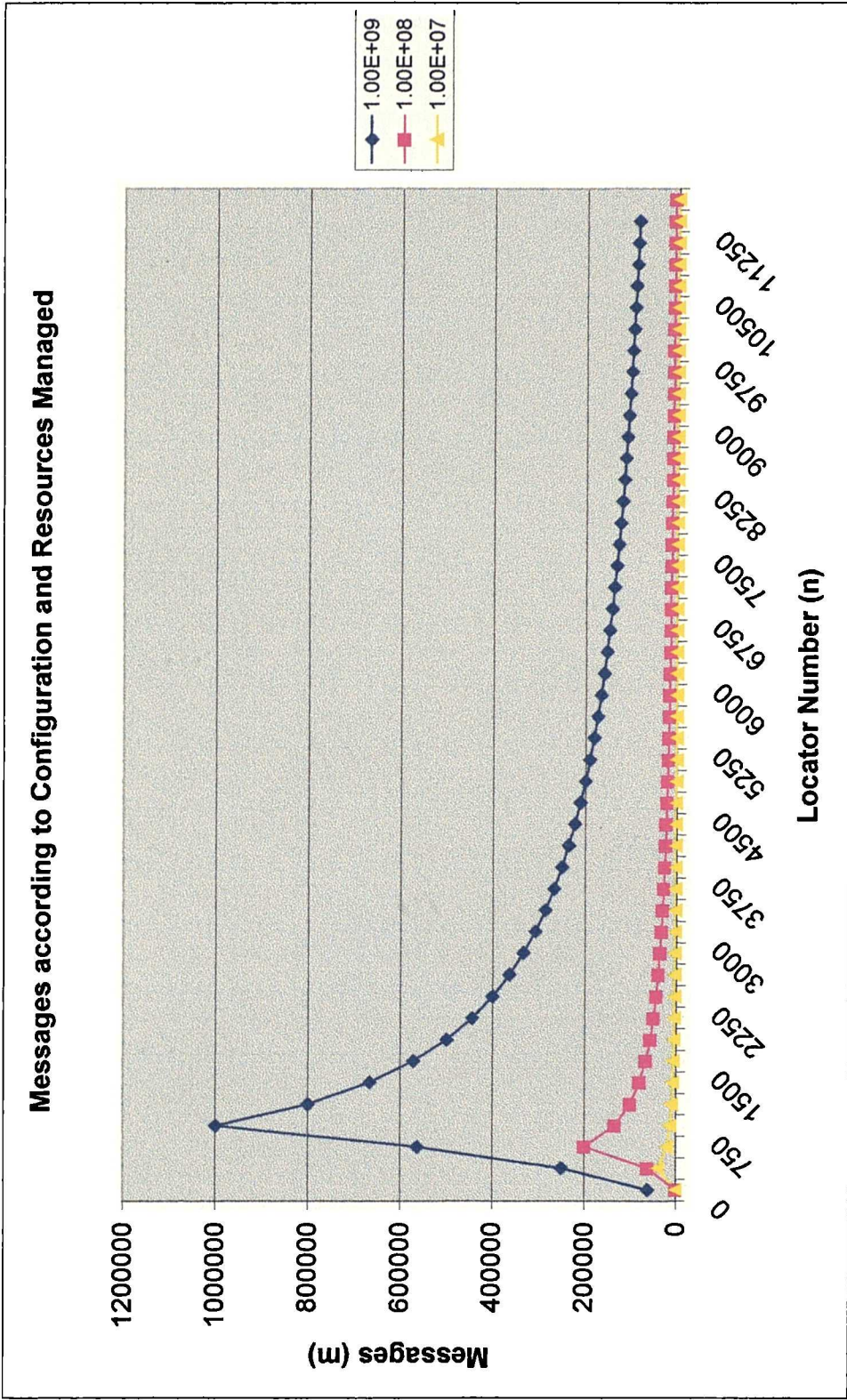


Figure 27 - Number of Messages Sent According to Configuration of RLS and Number of Resources Managed

That is, each Locator takes 1 hour 57 minutes to calculate which of its 1 million records need to migrate (keep in mind that all Locators perform this step roughly in parallel with one another).

The cost of the second step is dependent upon the number of records in the system and the number of Locators, with the number of messages generated increasing with the square of the number of Locators, until they reach the number of invalidated records, after which they decrease. This is illustrated in Figure 27, which shows the number of messages that are generated according to the number of Locators in the system, for different numbers of managed resources. Thus, for today's web with 10^9 resources, a 1,000 node system generates 1,000,000 messages (n^2). The number of records invalidated is therefore:

$$\frac{R}{n+1} = \frac{10^9}{1000} = 1,000,000 \text{ records}$$

Recall from section 6.3.4 that each record will be approximately 150 bytes in size; thus, total data transfer = 15MB. How fast this takes to complete depends entirely on the optimisation of the LCP, with messages sent in parallel substantially reducing the length of time required.

However, assuming the worst case scenario of linear operation (i.e. where only one of the one million messages is in transit at any one time), and a data transfer speed of 1.544 Mbps¹¹

¹¹ I.e. a standard T1 connection (Brebner, 1997); in contrast, the DNS uses 200Mbps of bandwidth (Kosters, 1999)

between Locators, the total time taken to transmit all one million messages (ignoring protocol overhead and converting bytes to bits) is:

$$\frac{1,000,000 \times 150 \times 8}{1,544,000} = 777.20 \text{ seconds, or 12 minutes 57 seconds.}$$

Thus, the total time taken to complete the addition of a new Locator is only 2 hours, 10 minutes, 4 seconds, which is entirely acceptable.

However, as can be seen from Figure 27, this is actually the *maximum* number of messages that could ever be generated for a system managing 10^9 resources. For example, if the number of Locators in the new configuration is 3,500, then the number of generated messages that must be broadcast comes down to approximately 285,714. Using the same figures as the previous example, the total time to transmit these messages is:

$$\frac{285,714 \times 150 \times 8}{1,544,000} = 222 \text{ seconds, = 3 mins, 42 seconds}$$

However, the total time taken for the first step would then increase. From Figure 26, the time taken for the Athlon 1100 MHz to calculate $t\#$ can be calculated as 0.0252 seconds (using the gradient of the slope). Thus, the time taken to complete step 1 would be:

$$\frac{10^9 \times 0.0252}{3500} = 7200 \text{ seconds = 2 hours.}$$

The total time would then come down to 2 hours, 3 mins 42 seconds, a saving of 6 minutes 22 seconds.

In fact, the two steps balance each other out, with the total time required to add a new Locator converging on 1 hour 59 minutes and 8 seconds regardless of the number of Locators in the system¹² (see Figure 28). The reason for this is that the total time taken (i.e. the sum of the two steps) converges on Rp (where p = time taken to hash one record with only one Locator in the system) as n increases. This can be illustrated by writing the sum of the two steps together:

$$t = \frac{Rt\#}{n+1} + \frac{b}{d} \cdot \frac{R}{n+1} \equiv \frac{Rt\#}{n+1} + \frac{Rb}{(n+1).d} \quad (i)$$

where t = total time, b = bit size of each record, d = data transfer speed in bits per second. But $t\#$ is linear with respect to n , as Figure 26 showed. That is, $t\# = pn + c$, where p = the gradient of the straight line (i.e the time taken to hash one record with only one Locator in the system) and c = the intercept, which is zero. Thus, $t\# = pn$, which, with $n+1$ Locators, gives:

$$t\# = p(n+1)$$

¹² Note that this figure is specific for the example system, which uses an Athlon 1100 MHz PC in a system managing one billion resources.

Substituting this value for $t\#$ in (i) gives:

$$t = \frac{Rp(n+1)}{n+1} + \frac{Rb}{(n+1)d} \equiv Rp + \frac{Rb}{(n+1)d}$$

Thus as $n \rightarrow \infty$, $t \rightarrow Rp$. As such, the time taken to add a Locator is bounded by Rp , and is therefore independent of n , the number of Locators in the system, thereby proving the system's scalability for configuration change.

Figure 28 demonstrates this result. The graph shows the time taken to add a new Locator for varying numbers of Locators (n) in a RLS that manages 1 billion resources. As can be seen, the graph converges on a time of 7,000 seconds as n increases. The timings are for a RR run on an Athlon 1100Mhz, and can be shown to converge on Rp as follows:

From Figure 26, the time taken to hash one record with only one Locator (p) can be calculated as:

$$p = \frac{t\#}{n} = \frac{0.007}{1000}$$

If $R = 10^9$, then

$$Rp = \frac{10^9 \times 0.007}{1000} = 7000$$

Time taken to add a new Locator according to Locator Number

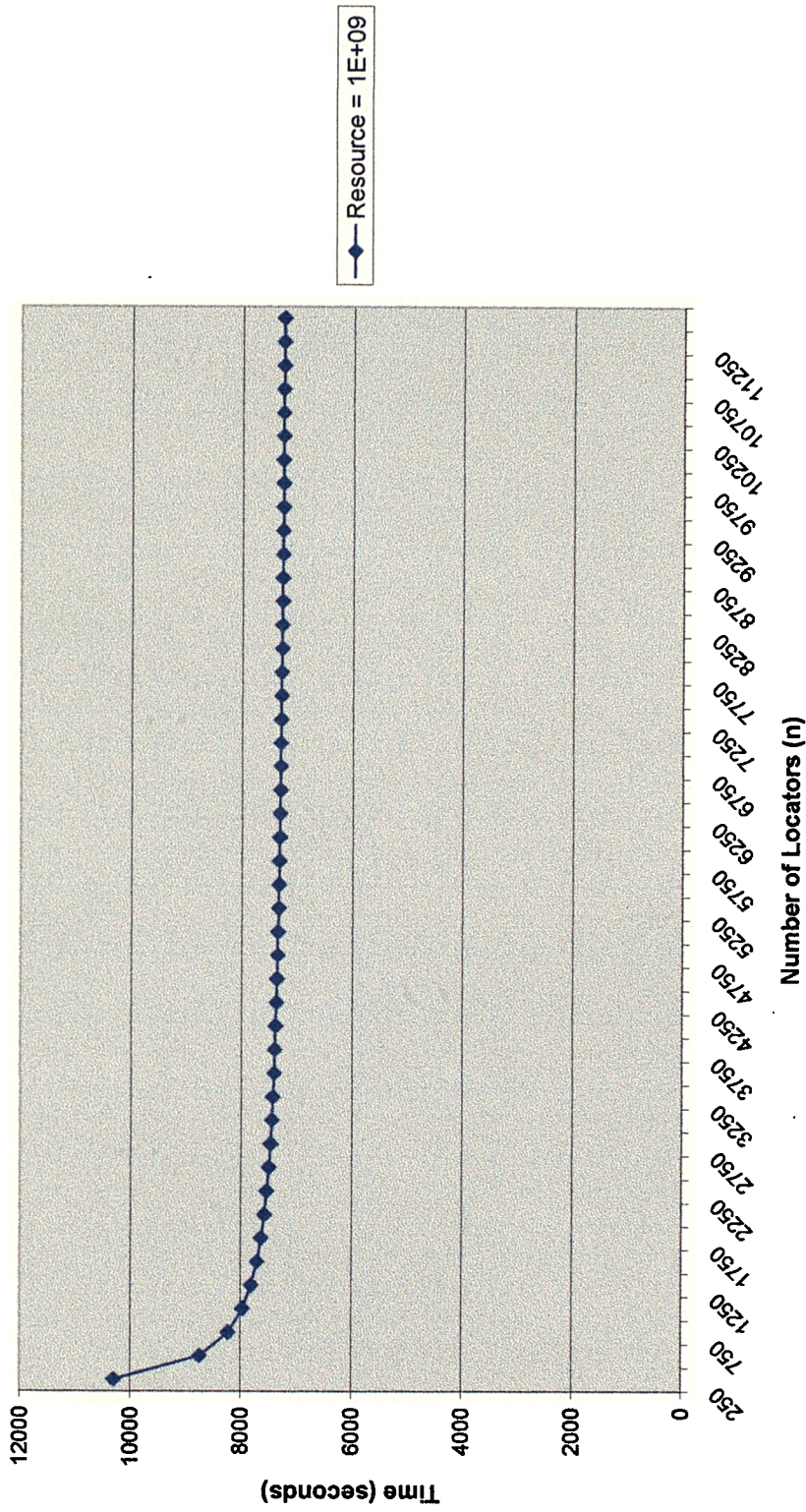


Figure 28 - Total time taken to add a new Locator according to Locator number

6.4.5.5 Performance Summary

These figures show that the design of the RLS is such that it can be deployed on today's web without experiencing any scalability problems. Furthermore, the system has been proven to be fully scalable according to the number of Locators, with the time taken to change configuration being independent of the number of Locators in the system. The sample figures used were obtained from a prototype system that used desktop PCs and non-optimised code, yet were still satisfactory for today's web, and showed that the prototype is capable of scaling beyond that. As such, a properly designed, highly optimised system that uses high performance machines should cope with a web many times the size that it is today.

6.4.6 Demonstrating New Services with the Prototype RLS

The RLS not only solves some of the flaws in the web, it also provides a platform upon which new services can be deployed. In this way, it extends the web's architecture by enabling it to provide more functionality than the existing architectural design can. As such, a number of small demonstration services have been developed that use the prototype RLS to demonstrate the enhanced functionality it provides. Specifically, the resource migration aspect of the RLS has been used to demonstrate:

- fault tolerance;
- load balancing;
- mobile agents.

These enhanced services work on top of the RLS, and operate transparently to any client, as the following sub-sections describe.

6.4.6.1 Load Balancing

The web provides little support for load balancing. The DNS can be used to provide a crude load balancing service, by returning different IP addresses for the same hostname, but this 'round robin' functionality (Albitz and Liu, 1997) only works at the host level, as browsers generally only perform a DNS query once for a whole web site.

In contrast, the RLS can provide a more sophisticated load balancing service at the resource level. Individual resources can be migrated dynamically according to the load they place on the server, thereby providing far more control than existing load balancing systems. Replicated resources will have the same name but different current locations. Assigning sequence orders to each replicated version allows a Locator to return the current URL of a different version for every HTTP request. This has been implemented in the prototype RLS, permitting resources to be placed on different servers according to media type, demand, or processing requirements, which is a level of management not supported by the DNS.

The management interface monitors the load on various resources. If the load gets too high, it migrates the resources to different predetermined web servers under its management, using the RMP. Once the load decreases, the resources are migrated back again. In this way, the interface acts as both client and Migration Manager.

6.4.6.2 *Fault Tolerance*

Mirroring a web site provides the web with a manual form of fault tolerance, but requires the resource owner to provide a different link to each of the various mirrors. If one of the mirrors falls over, the link to that mirror is broken, and the user must manually try another mirror through another link.

In contrast, the RLS can be used to automatically route around mirrors that have fallen over, and allow the resource owner to provide only one link. The management interface provides a demonstration of this (see Figure 29). Firstly, the user selects appropriate destinations for each resource that needs to be managed (each resource can be replicated onto a different server).

Once complete, the management interface automatically replicates selected resources onto selected web servers, using the RMP. The interface then monitors the resources. If it cannot access them, it updates the appropriate Locator, marking the inaccessible resource as inaccessible, and switching access over to the replicated resources. Once the system comes back up, the interface brings the Locator back to its original state. This demonstrates the usefulness of an *automatic* resource migration mechanism.

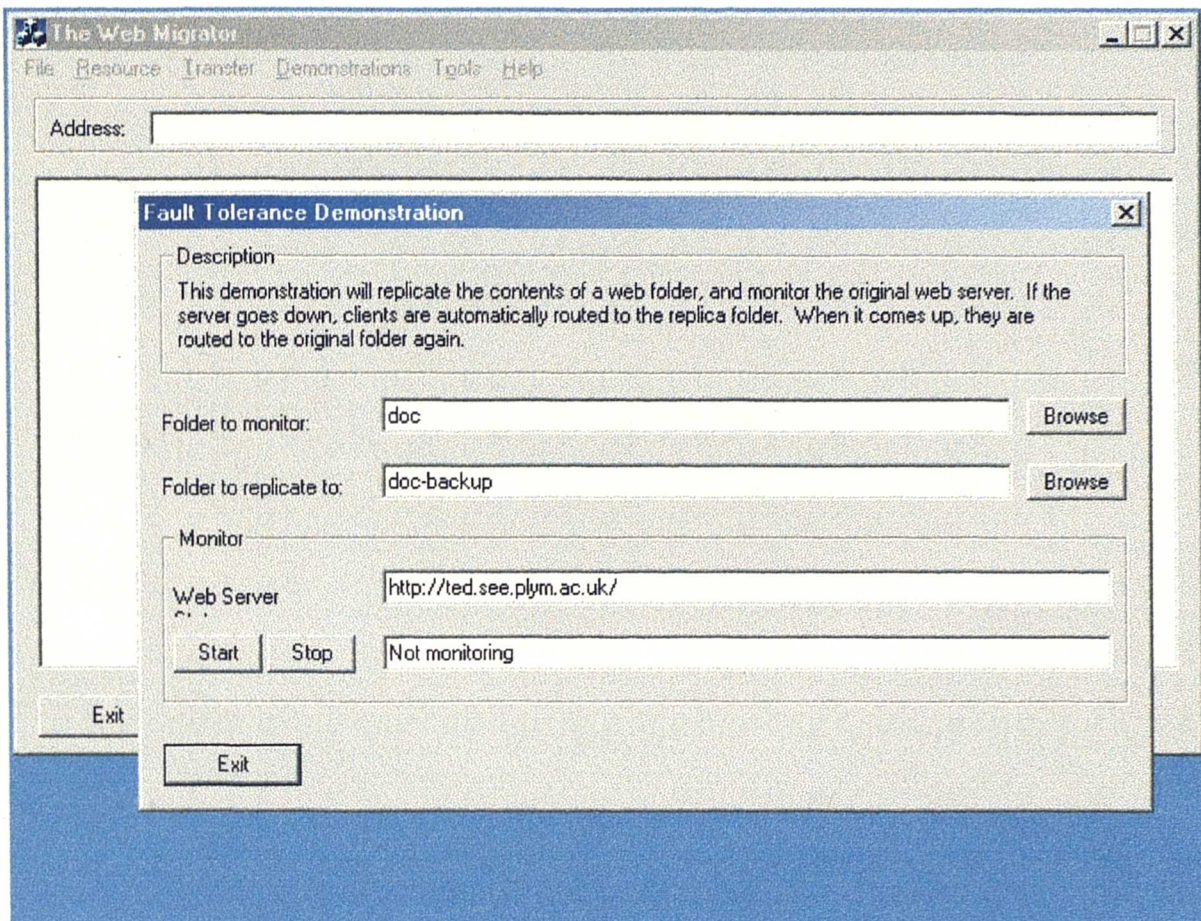


Figure 29 - Prototype Fault Tolerance Application of the RLS

6.4.6.3 Mobile Agents

Mobile agents are items of code that migrate across machines to perform their required task. However, there is currently no support for them on the web, largely because an agent's URL breaks each time it migrates. With the use of the RLS, however, any code can freely migrate, as it is treated by the RLS as just another resource.

The prototype system demonstrates mobile code by migrating a resource over to a random server every minute. The prototype uses four servers set up in the same room. The resource is a file containing a fragment of HTML, which is migrated to the same directory on the server as a mobile code-specific web page. This web page looks for the existence of the resource every 10 seconds; if it finds it, it reads the HTML contained within the resource and displays it. If not, it displays a blank screen (see Figure 30). Each server permanently displays the web page through a browser. As the management interface moves the resource across servers, the HTML contained within the resource is displayed on a different machine, providing a visual demonstration of migration.

The user can change the HTML in the mobile code at any time, no matter where the resource is currently located. To do this, the Management interface sends a HTTP GET to the appropriate Locator, which returns a HTTP *302 Found* response, with the current location of the response contained within the *location* header. The interface uses this location to issue a HTTP PUT command, updating the resource with the new HTML entered by the user, regardless of where the resource happens to reside. This demonstrates the transparency of the migration operation, as any client can download the resource and view its HTML contents at any time, regardless of which server it resides on, or where it is in the migration process, so long as it is connected to the proxy.

Although a trivial implementation, it demonstrates the ability of the RLS and RMP to act as a platform for mobile code and mobile agents. As such, a real mobile agent would contain more

functionality than a simple HTML file, and could be designed to contact a Migration Manager to migrate itself onto a different machine.

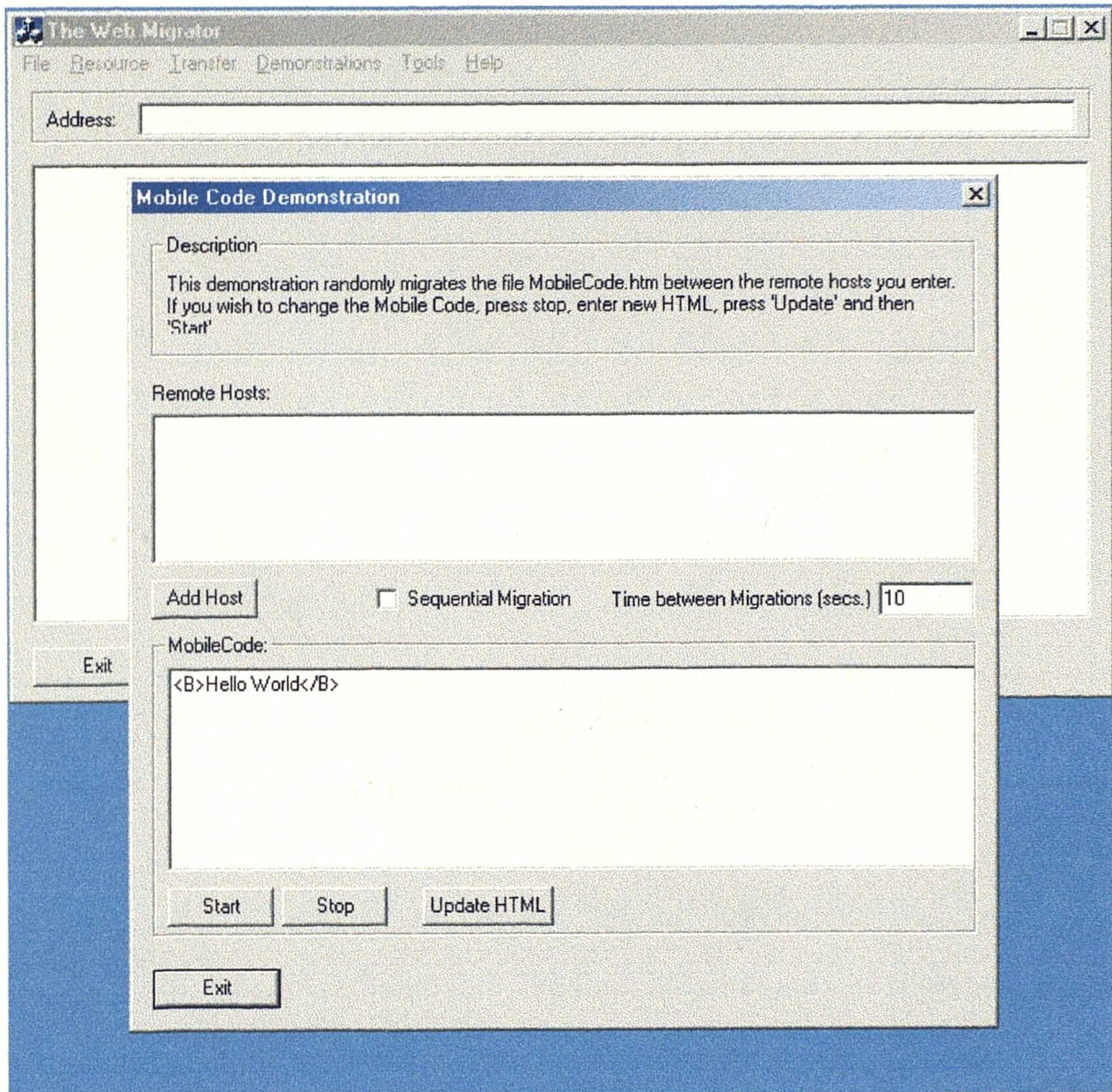


Figure 30 - Prototype Mobile Agent Demonstration

6.4.6.4 Other Enhanced Services

As well as those services that have been implemented to demonstrate the power of the RLS, many other services can be built upon its features that cannot be deployed using the web's existing architecture. Some examples include:

- *Web History*

By providing the web with a new temporal dimension, resources can be archived instead of lost. The temporal component of the RLS can enable them to be retrieved at any point in time. Extending this concept, however, third party services could provide temporal search engines, which dynamically return documents from a user's query based on a point in time. Thus, the user can see how many documents and of what quality existed at different points in time for the same query. The query might relate to an important news topic, or to a new technological or scientific breakthrough, and so the temporal search engine can be used to track its progress through time.

- *Security*

Recently, severe Distributed Denial of Service attacks have taken place against high profile web sites such as Yahoo and Amazon.com (McCullagh and Arent, 2000). The distributed denial of service attack is very difficult to counter, as it is hard to differentiate an attacker from a genuine user. However, with the RLS, resources can be dynamically mirrored and migrated across many different machines, thus dispersing the effectiveness of the attack. Various servers can be used to 'draw the fire' of the attack, while other servers continue to serve real users.

6.5 *Summary*

This chapter concludes the work that has been performed in order to validate the concepts that have been defined as part of the thesis. The chapter has presented the design and specification of the RLS and the RR, as well as a prototype implementation that demonstrates the power and flexibility of the system. In addition, the chapter has proven the scalability of the design, which has been tested with performance data from the prototype. More specifically, the chapter has presented:

- the design and specification of the Resource Migration Protocol (RMP);
- the design and specification of the Locator Control Protocol (LCP);
- a fully working prototype RLS and RR;
- demonstration applications of the RLS;
- performance figures of the system.

In addition, further validation of the system's design has been provided by the publication of a paper that describes the RLS in *Computer Networks* journal (Evans and Furnell, 2001). The RLS represents the culmination of the research, which, together with the OSN, has extended the existing state of the art, provided solutions to seemingly intractable problems, and will form the basis of much future research.

7. Conclusion

This chapter concludes the thesis by summarizing the work that has been achieved, including the new HOMINID model for managing information flow on the web, the Request Router and its associated algorithms and protocols, the Temporal URL, and the implementation of a prototype Resource Locator Service. The chapter concludes by discussing limitations of the research, and describing new areas of research that can be performed to enhance and refine the work further.

7.1 Achievements of the Research Programme

The research programme has met all of the objectives originally specified in chapter 1, with the exception of the link rot experiment, which had to be terminated prematurely prior to any conclusive results being obtained. However, a comprehensive literature search helped to quantify link rot, and new conceptual and practical work has been presented in a number of areas, as listed below.

1. The development of the HOMINID model for managing information flow, which solves the three identified web flaws without succumbing to the information management dichotomy. The HOMINID model comprises the Resource Locator Service, Temporal Referencing, and the Oracle Server Network, each of which extends the web's existing architecture without breaking it, and helps both the information provider and information user without censoring the web.
2. The design, development and testing of the Request Router, a novel node location system that transparently mediates between the web and a new distributed system. The Request Router is the foundation of the implementation of the RLS and OSN.
3. The design of temporal references, which add the dimension of time to the web without invalidating existing addressing schemes.

4. The design, specification, and development the RLS, complete with the implementation of a prototype to test the concept and measure its performance.
5. The design, specification, and development of the Resource Migration Protocol, which enables transparent resource migration across the web through an extension of existing web protocols.

Several papers relating to the research programme have been presented at refereed conferences and published in internationally recognized refereed journals, where the work received praise and recognition for its novel approach to solving the web's three flaws. In addition, the work on the RLS that involved the WebDAV protocol emerged as a result of participation in the IETF's WebDAV working group, and has led to the design and development of a novel application of WebDAV that enhances its functionality. In conclusion, it is believed that the research has made valid and useful contributions to the fields of distributed information management, distributed systems, and the World Wide Web.

7.2 Limitations of the Research

Despite having met the overall objectives of the research programme, and the functional success of the prototype, the work inevitably contains a number of limitations. The principal points are presented below.

1. The link rot experiment was terminated prematurely, leaving the quantification of link rot imprecise and out of date. Although the comprehensive literature review

subsequently provided adequate results, there is still no recent, comprehensive experimental data that is comparable with the experiments that have been performed regarding content lifetime.

2. Insufficient time and resources were available to further develop the OSN. The scope of the system could easily form another research programme in itself, including experiments, implementation and further design. However, the design and implementation of the RLS was the priority, and as the architectural design of the OSN relied on the success of the Request Router, the validation of the RLS's design applied equally to that of the OSN, at least from a functional perspective.
3. The RLS prototype was deployed and tested on a computer less powerful than if deployed on the web, and on a LAN rather than across the Internet, due to resource constraints. As such, the prototype was not run in a real world environment, which will have impacted the performance figures that were obtained. Despite this, however, the performance and scalability proved sufficient to validate its design such that it will scale to a system the size of the web.

7.3 Suggestions and Scope for Future Work

Throughout the thesis, areas where future work is possible or preferable have been identified, which could be conducted to build upon and enhance that undertaken within the project. These areas, together with new ones, are summarized below.

1. The link rot experiment can be remounted, but using a single HTTP HEAD request to determine the presence of a web server, rather than the ping method that was used in the original experiment. HTTP is deemed less ambiguous in its intent than ping, and so the security problems associated with the initial experiment should be removed.
2. The Oracle Server Network has not been fully specified or implemented. Future work will design its interface, and will determine the meta-data that needs to be stored. In addition, the OSN will be designed such that it can collate its information across resources, so that measurements can be made about the maximum and minimum values of the properties of resources and their content. In this way, the shape and structure of the web and its content can be accurately determined, enabling new services to be deployed that increase the web's functional value.
3. The design of the Locator could be extended such that a client could ask it to return a resource whose time of creation lies between a set of dates, rather than at a specific fixed date. The extra complexity this introduces into the design precluded it from becoming a feature of the RLS at this stage of its development, but it is a useful feature that should be considered for inclusion in the RLS's design in future work.
4. The resources that are managed by the RLS are simple, static resources with little or no intelligence. Existing distributed systems, such as CORBA or DCOM, use objects as their resources, which are full programmatic resources that have their own state and data. As such, future work should examine ways in which the RLS can be extended to manage

these intelligent objects, such that they can be transparently migrated across servers during execution. New developments on the web using XML and SOAP have provided the basis for the technology to achieve this, and the functionality of the RLS would provide new services to these existing distributed architectures.

5. The namespace of the RLS is left deliberately flat and unconstrained, but it is not known what effect this will have on the naming schemes used by resource owners. As such, future research will examine the merits of such a flat naming scheme, to determine whether or not any restrictions should be imposed on the RLS's namespace, and the exact nature of any restrictions should the research favour their introduction.

6. The current design of the RMP relies on a resource migration manager to co-ordinate the migration of a resource. However, there may be circumstances in which a client might wish to contact the manager remotely through the use of a new HTTP MIGRATE message. This would enable the client to direct its own migration, and could lead to third parties providing new migration services through an open, universal resource migration manager.

6. The RMP and LCP protocols make no allowance for security issues. As such, in the current design, anybody can alter a Locator's database, leading to corruption of its records. Future research should implement the ACL or at least HTTP digest authentication in order to safeguard the integrity of the RLS.

7. The design of the RMP and LCP is such that certain entities must wait for a response that may take a long time to arrive, due to the amount of processing that the machine sending the response has to do before the response can be sent. As such, future work should focus on providing disconnected operation, possibly through the use of a *defer-until* header, which would give a Locator time to complete its operation while providing a client application with the time in which it should check the Locator's progress. However, this functionality has been left out of the current design following discussions with Jim Whitehead, chair of the IETF's WebDAV working group, who advised that such disconnected operation would require some form of Internet-scale event notification technology, which does not yet exist (see Whitehead, 2000).

8. Temporal references were left out of the prototype. A future version should include these, and provide support for a temporal search engine, which would allow a user to query the web based on a topic and a range of times. Additionally, the future prototype should also include the OSN, to further realize its potential.

9. Further trials should place the prototype in a more realistic context, using larger machines to host the Locator software, an optimised Request Router, many Locators managing many resources, and distributed across a WAN rather than a LAN.

7.4 *The Future of the World Wide Web*

The World Wide Web is here to stay, and will remain part of our lives for many years to come. If it is to remain a useful resource, however, then its flaws need fixing, and they cannot

be fixed using methods that run counter to its philosophy. As such, the HOMINID model provides a complete solution to the information management flaws with which the thesis is concerned, providing a genuinely novel insight into the nature of information flow on the web. However, even if it does not become a new part of the web's architecture, it has contributed to the field in many ways:

- The HOMINID model is a novel means of managing information flow on the web. Its human-oriented perspective of information in a networked environment differs substantially from existing models, enabling it to more easily address the problems that emerge from the nature of information provision and consumption on the web. As such, it can be used in a wide range of applications, from enabling information providers to determine the best strategy for their marketing campaigns, to enabling browser designers to design better interfaces to the web.
- Request Routing is a novel approach to distributed systems, and can be used in any system that requires flexibility and performance while maintaining backwards compatibility.
- The hash routing algorithm used in the RR adapts the CARP algorithm in a way that removes the need for lookup tables of each node, and thus reduces the network overhead incurred by the algorithm. With the design of the LCP as a scalable protocol for controlling configuration change, the whole package can be used to lower the network overhead of any system that currently relies on CARP.

- The RMP is a generic automatic migration protocol that can be adopted by any migration mechanism, and used on the web by all entities without modification. The protocol also provides an impressive demonstration of what can be achieved with WebDAV, and an Internet draft will be submitted to the IETF as a contribution to Internet protocols.
- The TURL scheme is a novel extension to the URL that can be employed by any application on the web that requires a new temporal dimension. The specification of the scheme will also be submitted to the IETF as an Internet draft.
- The architecture of the RLS provides a novel approach to resource migration, improving the hyperlink's referential integrity without breaking the web's existing architecture. Although the design was developed specifically for the web, however, it should transfer well to other distributed systems, as the namespace is entirely generic, and web protocols are used simply to convey messages. As such, the design of the RLS can be seen as a novel resource migration mechanism for all types of information system, not just the web.
- The OSN improves the hyperlink's informational integrity, and enhances the richness of the web's links. The information that it stores can serve as a platform for many new services that require universal access to usage data on the web, as well as providing reliable metrics of the web and its users that can be used by third parties.

As such, the research work that has been completed for this PhD has contributed to many fields, and has provided new avenues for future research that will provide many more contributions in the future.

List of References

1. Akamai (1999), Akamai web site, <http://www.akamai.com/service/howitworks.html>
2. Arent, L. (1999), "Bidders High On Drugs.com?", Wired News, August 5th 1999, <http://www.wired.com/news/business/0,1367,21128,00.html>
3. Ashman, H. and Davis, H. (1998), W3C Panel: "Missing the 404: Link Integrity on the World Wide Web", in: The Seventh World Wide Web Conference, Brisbane, Australia, April 14-18 1998.
4. Barrett, T. (2000), "Internet comes of age with 30 millionth domain name" NetNames Web site, October 2000, <http://www.netnames.com/dnrs/netnames.client.Login>
5. Berners-Lee, T., Masinter, L. and M. McCahill (1994), "Uniform Resource Locators (URL)", RFC1738, December 1994
6. Berners-Lee, T. and Connolly, D (1995), "HyperText Markup Language Specification - 2.0", RFC1866, November 1995.
7. Berners-Lee, T. (1998) "Cool URIs Don't Change", W3C Web site, 1998, <http://www.w3.org/Provider/Style/URI>.

8. Berners-Lee, T., Fielding, R. and Masinter, L. (1998), "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, <http://www.ietf.org/rfc/rfc2396.txt>, August 1998.
9. Berners-Lee, T. and Fischetti, M. (1999), "Weaving the Web - the Past, Present and Future of the World Wide Web, by its Inventor", Orion Business Books, 1999.
10. Berst, J. (1998), "Search Sites' Shocking Secret", ZDNet Anchor Desk, August 17th, 1998, http://www.zdnet.com/anchordesk/story/story_2432.html
11. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen. H.F., Thatte, S. and Winer, D. (2000), "Simple Object Access Protocol (SOAP) 1.1", W3C Note, 8th May 2000, <http://www.w3.org/TR/SOAP/>
12. Braden, R. (1989), "Requirements for Internet Hosts - Communication layers, STD 3", RFC 1123, October 1989.
13. Bray, T., Paoli, J. and Sperberg-McQueen, C.M. (2000), "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, 6th October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>.
14. Brebner, G. (1997), "Computers in Communciation", 1997, McGraw Hill Publishing Company, Berkshire, England.

15. Brewington, B.E. and Cybenko, G. (2000), "Keeping up with the Changing Web", IEEE Computer, p 52-58, May 2000
16. Brin, S and Page, L. (1998), "The Anatomy of a Large-Scale Hypertextual Web Search Engine", in: Proc. 7th International World Wide Web Conference Brisbane, Australia, 14-18th April, 1998
17. Briscoe, R. J. (1997), "Distributed Objects on the Web", BT Technology Journal, Vol.15 No.2, April 1997, pp158.
18. Broder, A.Z., Glassman, S.C., Manasse, M.S, and Zweig, G. (1997), "Syntactic Clustering of the Web", In: Proceedings of the 6th International World Wide Web Conference, pp. 391-404, 1997
19. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A. and Wiener, J. (2000), "Graph Structure in the Web" In: Proceedings of the 9th International World Wide Web Conference, Amsterdam, The Netherlands, May 2000
20. Butler, D. (2000), "Souped-Up Search Engines", in: Nature, No. 405, pp112-115, 11 May 2000, http://www.nature.com/cgi-taf/DynaPage.taf?file=/nature/journal/v405/n6783_full/405112a0_fs.html&_UserReference=C0A804EC46B4E67A8CDD728163CF3A6D6B22

21. Catledge, L.D. and Pitkow, J.E. (1995), "Characterizing Browsing Strategies in the World-Wide Web", in: Proceedings of the Third International World Wide Web Conference, Darmstadt, Germany, April 1995
22. Cerf, V. and Kahn, R. (1974), "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.
23. Chakrabarti, S., van den Berg, M. and Dom, B. (1999a), "Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery", in: Proceedings of the 8th International World Wide Web Conference, 1999.
24. Chakrabarti, S., Dom, B.E., Gibson, D., Kleinberg, J., Kumar, R., Raghavan, P., Rajagopalan S. and Tomkins A. (1999b), "Mining the Link Structure of the World Wide Web", IEEE Computer, 32(8), August 1999.
25. Chankhunthod, A., Danzig, P.B., Neerdaels, C., Schwartz, M.F. and Worrell, K.J. (1995), Object Lifetimes, in A Hierarchical Internet Object Cache, 1995.
<http://excalibur.usc.edu/cache-html/cache.html>.
26. Cockburn, A. and Greenberg, S. (1999), "Issues of Page Representation and Organization in Web Browser's Revisitation Tools" in: Proceedings of the OZCHI'99 Australian Conferences on Human Computer Interaction, November 28-30, Wagga Wagga, Australia.

27. Cohen, J., Hopman, A., Goland, Y., Valloppillil, V., Leach, P. and Lawrence, S. (1998), "Don't Go Postal - An Argument Against Improperly Overloading the HTTP POST Method", Internet draft, draft-cohen-http-ext-postal-00, 1998
28. Concise Oxford Dictionary (COD) (1990), "Concise Oxford Dictionary", 1990, R.E. Allen (Ed.), Clarendon Press, Oxford
29. Cox, B. (1999), "Bonzi Software Banner Leads NetRatings Weekly List", InternetNews.com, March 15th, 1999,
http://www.internetnews.com/IAR/article/0,,12_80021,00.html
30. Cuenca, P., Sosa, V., Romero, J. and Hernanz, I. (1999), "Lessons Learned from the Early Adoption of URNs in an Intranet Environment", The 9th Annual Conference of the Internet Society, INET 99, San Jose, CA,
http://www.isoc.org/inet99/proceedings/4m/4m_2.htm, 1999
31. CyberMetrics (2000) Homepage of the International Journal of Scientometrics, Informetrics and Bibliometrics, October 2000,
<http://www.cindoc.csic.es/cybermetrics/links08.html>
32. Daniel, R. and Mealling., M, (1997), "Resolution of Uniform Resource Identifiers using the Domain Name System", RFC2168, June 1997.

33. DeRose, S, Maler, E. and Daniel, R. (2001), "XML Pointer Language (Xpointer) version 1.0", World Wide Web Consortium Last Call Working Draft, 8th January, 2001, WD-xptr-19980303, 3rd March 1998, <http://www.w3.org/TR/2001/WD-xptr-20010108/>
34. DeRose, S, Maler, E. and Orchard, D. (2000), "XML Linking Language (Xlink) - version 1.0", World Wide Web Consortium Proposed Recommendation, 20th December 2000, <http://www.w3.org/TR/2000/PR-xlink-20001220/>
35. Devlin, K. (1991) "Information and Logic", Cambridge University Press, 1991
36. DOLMEN (1995), "Service Machine Development for an Open Long-term Mobile and Fixed Network Environment - Technical Annex", ACTS DOLMEN, 1995
37. DOLMEN (1996a), "Evaluation of Service Architecture Frameworks", G. Bruno, ACTS DOLMEN Deliverable ASD1, 28th June 1996
38. DOLMEN (1996b), "Evaluation of Current Communication Technology in Hypermedia Information Browsing", Raatikainen, K., ACTS DOLMEN deliverable TAD3, 1996
39. DOLMEN (1997), "Implementation of an Enhanced Distributed Processing Platform for DOLMEN", Huynh, N., ACTS DOLMEN deliverable MPD3, 15th April 1997

40. DomainStats (2000), DomainStats web site, <http://www.domainstats.com/>, October 2000
41. DotCom (2000), DotCom web site, <http://www.dotcom.com/facts/quickstats.html>
42. Douglis, F., Feldmann, A. and Krisnamurthy, B. (1997), "Rate of Change and other Metrics: a Live Study of the World Wide Web", In: Proceedings of USENIX Symposium on Internet Technology, and Systems, Monterey, CA, pp. 147-158, December 1997. 12
43. Dublin Core Working Group (DublinCoreWG) (1999), "Dublin Core Metadata Element Set, Version 1.1: Reference Description", 2nd July, 1999, <http://purl.org/dc/documents/rec-dces-19990702.htm>
44. Esposito, D. (1999), "ADSI Overview", Microsoft Internet Developer Journal, May 1999, <http://www.microsoft.com/Mind/0599/cutting/cutting0599.htm>
45. Evans, M.P., Phippen, A.D., Mueller, G., Furnell, S.M., Sanders, P.W. and Reynolds, P.L. (1999) "Strategies for Content Migration on the World Wide Web" Internet Research, vol. 9, no. 1, 1999. pp25-34.
46. Evans, M.P. and Furnell, S.M. (2000), "Internet-based security incidents and the potential for false alarms", Internet Research, vol. 10, no. 3: 238-245. , 2000

47. Evans, M.P. and Furnell, S.M. (2001), "The Resource Locator Service: Fixing a Flaw in the Web", to appear in *Computer Networks Journal - The International Journal of Computer and Telecommunications Networking*, Elsevier Science
48. Feldman, S. (1998), "The Internet Search-Off", *The Searcher: The Magazine for Database Professionals*, February 1998,
<http://www.infoday.com/searcher/feb98/story1.htm>
49. Fielding, R, Gettys, J, Mogul, J.C., Nielsen, H.F., Masinter, L, Leach, P., Berners-Lee, T. (1999), *HyperText Transfer Protocol - HTTP/1.1*, RFC 2616, June 1999.
50. Fielding, R.T., (1996), "Fielding on MOVE & COPY", Discussion in WebDAV working group, July to September 1996, <http://lists.w3.org/Archives/Public/w3c-dist-auth/1996JulSep/0045.htm>
51. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L. (1999), "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, <http://www.rfc-editor.org/rfc/rfc2617.txt>
52. Freed, N. and Borenstein, N. (1996), "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996

53. Gilder, G. (1993), "Metcalfe's Law and Legacy", Forbes ASAP Magazine, 13th September, 1993
54. Gillet, S.E. and Kapor, M. (1997), "The Self-Governing Internet: Coordination by Design", in: Coordination of the Internet, ed. B. Kahin and J. Keller, MIT Press, 1997
55. GlobalReach (2000), Global Reach homepage, Global Reach, <http://www.gltreach.com/globstats/index.php3>, October 2000
56. Goland, Y., Whitehead, E.J., Faizi, A., Carter, S.R. and Jenson, D. (1999), "HTTP Extension for Distributed Authoring - WebDAV", RFC 2518, February 1999.
57. Graphic, Visualization and Usability Centre (GVU) (1997), "7th WWW User Survey", April 1997 (an archive of GVU's web surveys can be found at http://www.cc.gatech.edu/user_surveys/)
58. Graphic, Visualization and Usability Centre (GVU) (1998), "9th WWW User Survey", April 1998
59. Harris, C. (2000), "LinkGuard - Intelligent Link Management", White Paper, Link Guard web site, <http://www.linkguard.com/utills/downloads/wp/whitepaper.pdf>
60. Heery, D. (1996), "Review of Metadata Formats", Program, Vol. 30, No. 4, October 1996, pp. 345-373

61. Henziger, M.R., Heydon, A., Mitzenmacher, M. and Njork, M. (1999), "Measuring Index Quality using Random Walks on the Web", In Eighth International World Wide Web Conference, pages 213-225, Elsevier Science B.V., May 1999
62. Higgins, M. (1999), "Meta-Information, and Time: Factors in Human Decision making", Journal of the American Society for Information Science, 50(2): 132-139, 1999
63. Hochheiser, H. and Schneiderman, B. (1999), "Understanding Patterns of User Visits to Web Sites: Interactive Starfield Visualizations of WWW Log Data", In: Proceedings of ASIS '99, 1999.
64. Holtman, K. and Mutz, A. (1998), "Transparent Content Negotiation in HTTP", RFC2295, March 1998
65. Huberman, B.A., Pirolli, P.L.T., Pitkow, J.E. and Lukse, R.M. (1998), "Strong Regularities in World Wide Surfing", Science, Vol.280, 3rd April 1998
66. Iannella, R., Sue, H. and Leong, D. (1996), "BURNS: Basic URN Service Resolution for the Internet", in: Proceedings of the Asia-Pacific World Wide Web Conference, Beijing & Hog Kong, 1996,
http://www.dstc.edu.au/Research/Research/Resource_Discovery/publications/apweb96/index.html

67. Ingham, D, Caughey, S and Little, M. (1996), "Fixing the 'Broken-Link' Problem: The W3Objects Approach", in: The Fifth International World Wide Web Conference, Paris, France, May 6-10 1996.
68. Ingham, D., Little, M., Caughey, S. and Shrivastava, S. (1995), "W3Objects: Bringing Object-Oriented Technology to the Web", in: Proceedings of the 4th International WWW Conference, Boston, December 1995,
<http://www.w3.org/pub/Conferences/WWW4/Papers2/141>
69. Inktomi Corporation (1996), "The Inktomi Technology Behind HotBot - A White Paper", 1996, <http://www.inktomi.com/products/search/clustered.html>
70. Inktomi Corporation (1999), Inktomi and NEC Research institute,
<http://www.inktomi.com/webmap/>.
71. Internet Corporation for Assigned Names and Numbers (ICANN) (1999), "Uniform Domain Name Resolution Policy", <http://www.icann.org/udrp/udrp-policy-24oct99.htm>, October 24th 1999
72. ISO/IEC (1993) "Draft Recommendation X.901: Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to use", ISO/IEC, 30th August 1993.

73. Kahle, B. (1996), "Archiving the Internet", First draft of Kahle 1997, submitted to Scientific American in 4th November 1996, http://www.archive.org/sciam_article.html
74. Kahle, B. (1997), "Preserving the Internet", Scientific American, March 1997, <http://www.sciam.com/0397issue/0397kahle.html>
75. Kahle, B. (1999), Personal communication (see Appendix C), July 15th 1999
76. Kawai, E., Osuga, K., Chinien, K. and Yamaguchi, S. (2000), "Duplicated Hash Routing: A Robust Algorithm for a Distributed WWW Cache System", in: IEICE Trans. Inf. & Syst., Vol.E83-D, No.5, May 2000.
77. Kleinberg, J.M. (1998), "Authoritative Sources in a Hyperlinked Environment", Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, pp. 668-677, January 1998.
78. Klyne, G. (1999), "Protocol-Independent Content Negotiation Framework", RFC 2703, September 1999.
79. Knuth, D. (1998), "The Art of Computer Programming - Volume III Sorting and Searching", 2nd Edition, Addison Wesley Longman, 1998

80. Koehler, W. (1999), "An Analysis of Web Page and Web Site Constancy and Permanence", *Journal of the American Society for Information Science*, 50(2);162-180, 1999
81. Kusters, M. (1999), "Massive Scale Name Management: Lessons Learned from the .COM Namespace", TWIST 99 conference, University of California, Irvine, California, USA, August 1999.
82. Lagoze, C. (1997), "From Static to Dynamic Surrogates - Resource Discovery in the Digital Age", *D-Lib Magazine*, June 1997, <http://mirrored.ukoln.ac.uk/lis-journals/dlib/dlib/dlib/june97/06lagoze.html>
83. Lagoze, C. and Fielding, D. (1998), "Defining Collections in Distributed Digital Libraries", *D-Lib Magazine*, ISSN 1082-9873, November 1998, <http://mirrored.ukoln.ac.uk/lis-jouranls/dlib/dlib/dlib/november98/lagoze/11lagoze.html>
84. Lassila, O. (1997), "Introduction to RDF Meta-Data", W3C NOTE 1997-11-13, 13th November 1997, <http://www.w3.org/TR/NOTE-rdf-simple-intro>
85. Lassila, O. and Swick, R.R. (1998), "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, 22ndFebruary, 1999, RD-rdf-syntax-19980819, <http://www.w3.org/TR/REC-rdf-syntax/>

86. Lawrence, S. and Giles, C.L. (1999), "Accessibility of Information on the Web", Nature, Vol.400, 8 July 1999, pp107-109.
87. Lawrence, S., and Giles, C.L. (1998), "Inquirus, the NECI Meta-Search Engine", In: Proceedings of the Seventh International World Wide Web Conference, Brisbane, Australia, 14-18 April 1998, pages 95-105.
<http://www7.scu.edu.au/programme/fullpapers/1906/com1906.htm>
88. Lawrence, S., Pennock, D.M., Flake, G.W., Krovetz, R., Coetzee, F.M., Glover, E., Nielsen, F.A., Kruger, A. and Giles, C.L. (2001), "Persistence of Web References in Scientific Research", IEEE Computer, p26-31, February 2001.
89. Levinson, P. (1997) "The Soft Edge - A Natural History and Future of the Information Revolution", Routledge, 1997
90. Mackay, D. (2001), "Information Theory, Inference, and Learning Algorithms", Cambridge University Press, Cambridge, UK. See also
<http://wol.ra.phy.cam.ac.uk/mackay/itprnn/book.html>
91. McCullagh, D. (1999), "Domain Name List is Dwindling", Wired News, April 14 1999, <http://www.wired.com/news/technology/0,1282,19117,00.html>

92. McCullagh, D. and L. Arent (2000), "A Frenzy of Hacking Attacks", Wired News, 9 February 2000. <http://www.wired.com/news/print/0,1294,34234,00.html>.
93. McNamara, P. (2000), "Guarding against Broken Links", Network World Fusion News Article, <http://www.nwfusion.com/columnists/2000/0515netbuzz.html>, 15th May 2000
94. Metcalfe, R. (1996), "Computer Laws galore, but one is holding back the information age", InfoWorld article, May 6th 1996, <http://www.infoworld.com/cgi-bin/displayNew.pl?metcalfe/bm050696.htm>
95. Microsoft (1997a), "MS Active Directory Service Interface (ADSI)" <http://www.microsoft.com/technet/winnt/winntas/technote/adsiwsp.asp>, February 1997.
96. Microsoft (1997b), "Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0", Microsoft Corporation, <http://msdn.microsoft.com/library/backgmd/html/carp.htm>
97. Microsoft (2000a), "Windows Internet Name Service and Broadcast Name Resolution", Microsoft, 2000, <http://msdn.microsoft.com/library/default.asp?URL=/library/winresource/dnwinnt/S763A.HTM>
98. Microsoft (2000b), "About the OLE DB Provider for Internet Publishing", Microsoft, 2000. http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/ipubsdk/ipubbyb0_797m.htm
99. Microsoft (2001). Microsoft DotNet Web Site, <http://msdn.microsoft.com/net/>

100. Mitchell, D., Bradner, S. and Claffy, K. (1996), "In whose Domain: Name Service in Adolescence", Information Infrastructure project workshop on Co-ordination and Administration of the Internet JFK School, 8-10 September, 1996,
<http://www.caida.org/outreach/papers/dnssence>
101. Mockapetris, P. (1987a), "Domain names - concepts and facilities," RFC1034, November 1987, <http://www.ietf.org/rfc/rfc1034.txt>.
102. Mockapetris, P. (1987b), "Domain names - implementation and specification", RFC1035, November 1987, <http://www.ietf.org/rfc/rfc1035.txt>.
103. Moore, K. (1996), "Location-Independent URLs or URNs Considered Harmful", Internet Draft, draft-ietf-uri-urns-harmful-00.txt, 1996
104. Nielsen, H.F. and D. LaLiberte (1999), "Editing the Web: Detecting the Lost Update Problem Using Unreserved Checkout", W3C NOTE, May 10, 1999,
<http://www.w3.org/1999/04/Editing/>
105. Nielsen, H.F., Leach, P. and Lawrence, S. (2000), "An HTTP Extension Framework", RFC 2774, February 2000.

106. Notess, G. (2000a), "Search Engine Statistics Web site",
<http://www.notess.com/search/stats/sizeest.shtml>, 2000.
107. Notess, G. (2000b), Search Engine Showdown report, February 21, 2000,
<http://searchengineshowdown.com/stats/dead.shtml>
108. Object Management Group (OMG) (2000), "Interoperable Naming Service Specification", Object Management Group, Document number: formal/2000-11-01,
http://www.omg.org/technology/documents/formal/naming_service.htm, November 2000
109. Ohto, H. and Hjelm, J (1999), "CC/PP Exchange Protocol Based on HTTP Extension Framework", W3C Note, 24th June 1999, <http://www.w3.org/TR/NOTE-CCPPexchange.htm>
110. OMG (1995), "The Common Object Request Broker: Architecture and Specification, Revision 2.0", Object Management Group, 1995
111. Pallmann, D. (1999), Pallmann, D., "Programming Bots, Spiders and Intelligent Agents in Microsoft Visual C++", Microsoft Press, Redmond, Washington, USA, 1999.

112. Park, K., Kim, G. and Crovella, M. (1997), "On the Effect of Traffic Self-Similarity on Network Performance", in Proceedings of the 1997 SPIE International Conference on Performance and Control of Network Systems, November 1997
113. Pearson, S. (2000), "Hype or Hypertext? A Plan for the Law Review to move into the 21st Century", <http://www.cc.utah.edu/~sgp5837/pearson.htm>
114. Pitkow, J, and Recker, M. (1994), "Integrating Bottom-Up and Top-Down Analysis for Intelligent Hypertext", Intelligent Hypertext Workshop, Third International Conference on information and Knowledge Management, National Institute of StandardTechnology, December 12th 1994
115. Pitkow, J. E. (1998), "Summary of WWW characterizations", Computer Networks and ISDN Systems, vol. 30, no. 5, pp. 551-558, 1998
116. Pitkow, J.E. (1997), "In Search of Reliable Usage Data on the WWW", The Sixth International World Wide Web Conference, pages 451-463, Santa Clara, CA, 1997.
117. Pitkow, J.E. and Jones, R.K. (1996), "Supporting the Web: a Distributed Hyperlink Database System", in: The Fifth International World Wide Web Conference, Paris, France, May 6-10 1996.

118. Postel, J. (1981a), "Transmission Control Protocol - DARPA Internet Program Protocol Specification", Postel, J. (ed.), RFC793, September 1981.
119. Postel, J. (1981b), "Internet Protocol - DARPA Internet Program Protocol Specification", Postel, J. (ed.), RFC791, September 1981.
120. Postel, J. (1996), "Media Type Registration Procedure", RFC 1590, November 1996.
121. Raatikainen, K., Dede, A. and Koskimies, O. (1998), "Internet browsing on OSAM platform", In: Intelligence in services and networks: technology for ubiquitous telecom services. Proc. 5th International Conference on Intelligence in Services and Networks, IS&N '98, Antwerp, Belgium, May 25-28, 1998. Berlin, Springer-Verlag, 1998 (Lecture notes in computer science vol. 1430) pp. 261-272
122. Reddy, S., Lowry, D., Reddy, S., Henderson, R., Davis, J. and Babich, A., (1999), "DAV Searching and Locating", Internet Draft draft-dasl-protocol-01, <http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html>, July 1999.
123. Ross, K. (1997), "Hash Routing for Collections of Shared Web Caches", IEEE Network Magazine, 11, 7:37--44, Nov-Dec 1997.
124. Schwartz, C. (1998), "Web Search Engines", Journal of the American Society for Information Science, 49(1), p973-982, 1998

125. Sedlar, E and G. Clemm (2000), "Access Control Extensions to WebDAV", Internet Draft draft-ietf-webdav-acl-01, April 28 2000,
<http://www.webdav.org/acl/protocol/draft-ietf-webdav-acl-01.htm>.
126. Shafer, K., Weibel, S., Jul, E. and Fausey, J. (1996), "Introduction to Persistent Uniform Resource Locators", in: Proceedings of INET96, Montreal, Canada, 24-28 June 1996.
127. Slein, J.A., Vitali, F., Whitehead, J. and Durand, D. (1998), "Requirements for a Distributed Authoring and Versioning Protocol for the World Wide Web", RFC 2291, February 1998, <http://www.rfc-editor.org/rfc/rfc2291.txt>
128. Sollins, K. (1998), "Architectural Principles of Uniform Resource Name Resolution", RFC 2276, January 1998.
129. Sollins, K. and Masinter, L. (1994), "Functional Requirements for Uniform Resource Names", RFC 1737, December 1994
130. SRI Research (2000), "How People Use the Internet", 17th February 2000,
<http://www.sriresearch.com/press/pr20000217.htm>

131. Stallings, W. (1991), Stallings, W, "Data and Computer Communications", 4th Edition, Macmillan Publishing Company, New York, 1991.
132. Sullivan, D. (2000a), "SearchEngineWatch" report, July 7th 2000,
<http://www.searchenginewatch.com/reports/directories.html>
133. Sullivan, D. (2000c), "Search Engine Watch",
<http://www.searchenginewatch.com/reports/perday.html>
134. Sullivan, D. (2000d) "Search Engine Watch",
<http://www.searchenginewatch.com/webmasters/features.html>
135. Sullivan, T. (2000b), "All Things Web", <http://www.pantos.org/atw/35654.html>, 2000.
136. Sun, S.X. and Lannom, L., (2000), "The Handle System: A Persistent Global Name Service - Overview and Syntax", Internet-draft, February 2000,
<http://www.ietf.org/internet-drafts/draft-sun-handle-system-04.txt>
137. Tauscher, L. and Greenberg, S. (1997), "Revisitation Patterns in World Wide Web Navigation", Conference on Human Factors in Computer Systems, Atlanta, Georgia, March 22-27, 1997.

138. Thaler, D.G. and Ravishankar, C.V. (1998), "Using Name Based Mappings to Increase Hit Rates", IEEE/ACM Transactions on Networking, 6(1):1-14, February 1998.
139. TINA (1994) "Overall Concepts and principles of TINA Version 1.0", Chapman, M. and Montesi, S., TINA-C Deliverable, 17th February 1995
140. University of Michigan (1995), University of Michigan's LDAP FAQ, 1995,
<http://www.umich.edu/~dirsvcs/ldap/doc/guides/slapd/1.html#RTFToC1>
141. Valloppillil, V. and Ross, K. (1998), "Cache Array Routing Protocol v1.0", Internet Draft, draft-vinod-carp-v1-03, 26th February 1998,
<http://www.microsoft.com/proxy/guide/CarpSpec.asp?A=2&B=3>
142. Vixie, P., Thomson, S., Rekhter, Y. and Bound, J. (1997), "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997
143. W3C (1992) World Wide Web Consortium web site, <http://www.w3.org/WWW/>, 1992
144. Whitehead, E.J. and Wiggins, M. (1998), "WebDAV: IETF Standard for Collaborative Authoring on the Web", IEEE Internet Computing: Software Engineering over the Internet, 2(5): 34-40, September-October 1998

List of References

145. Whitehead, E.J. (2000), Personal Communication via IETF's WebDAV working group.
146. Windrum, P. (1999), "The Collective Invention of the World Wide Web", the Colline Report, Prepared for DGXII, European Commission, 1999
147. Yeong, W., Howes, T. and Kille, S. (1995), "The Lightweight Directory Access Protocol", RFC 1777, <ftp://ftp.isi.edu/in-notes/rfc1777.txt>, March 1995.

APPENDIX A

CORE COMPONENTS OF THE HOMINID MODEL

The core components of the HOMINID model that were developed in chapter 4 are presented in the following table for reference.

Concept	Problem Solved	Description
Temporal Reference	<ul style="list-style-type: none">• Invalid hyperlinks due to content change• Lost History	The Temporal Reference binds a resource and its content together as one atomic unit, and locates that unit in time and space. Should any component of this unit change, it becomes a new unit, and must receive a new temporal reference.
Resource Locator Service	<ul style="list-style-type: none">• Link Rot• Shrinking Namespace• Automatic, transparent resource migration	The RLS is functionally equivalent to the DNS, but does not constrain the namespace. Its default namespace is the temporal reference, which enables it to locate a resource across time and space. The RLS also provides a transparent resource migration mechanism that can enable a resource to be migrated remotely.

The Core Components of the HOMINID Model

Concept	Problem Solved	Description
Oracle Server	<ul style="list-style-type: none"> • Increasing Noise • Ineffective Browsing caused by deceptive hyperlinks • Resolves the Information Management Dichotomy • Measuring the Web 	<p>The Oracle Server provides universal access to characteristic infons and navigational infons about the resources on the web and the way in which they are used. In this way, it can measure the resource and its referring hyperlinks, and provides the user with information from the Oracle situation rather than from the resource owner's deceptive situation, thus maintaining the hyperlink's informational integrity. As such, the Oracle Server can alert the user to deceptive strategies and help them to make more informed browsing choices. This reduces the noise in the web without requiring the censorship of its information, and so resolves the information management dichotomy.</p>

The Core Components of the HOMINID Model (cont.)

APPENDIX B

LIST OF ABBREVIATIONS

CORBA	Common Object Request Broker Architecture
CARP	Cache Array Routing Protocol
DNS	Domain Name System
DOLMEN	Service Machine Development for an Open Long-Term Mobile and Fixed Network Environment
HOMINID	Human-Oriented model for Managing Information flow on the web
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ISP	Internet Service Provider
LCP	Locator Control Protocol
OSAM	Open Service Architecture for Mobiles
OSN	Oracle Server Network
OSP	Online Service Provider
QoS	Quality of Service
RDF	Resource Description Framework
RLS	Resource Locator Service
RMP	Resource Migration Protocol
RR	Request Router

TCP/IP	Transmission Control Protocol/Internet Protocol
TLD	Top-Level Domain name
TURL	Temporal Uniform Resource Locator
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Identifier
WebDAV	Web Distributed Authoring and Versioning
XML	Extensible Markup Language

APPENDIX C

PERSONAL COMMUNICATIONS

The following represents personal communication with Brewster Kahle, CEO of the Alexa archive. The communication requested clarification of the reference used in Kahle's Scientific American article regarding the lifetime of a resource on the web (see p57).

Mike Evans

From: Brewster Kahle [brewster@alexa.com]
Sent: Thursday, July 15, 1999 9:31 PM
To: 'Mike Evans'
Subject: RE: Where did 44 days come from?

It is that danzig paper that that number came from. An internal stat for us is that 6% of HTML pages changed in 3 months (this does not mean that they did not disappear). This argues that the 44 days is no longer valid.

-brewster

>-----Original Message-----

>From: Mike Evans [mailto:Mike.Evans@jack.see.plym.ac.uk]
>Sent: Thursday, July 15, 1999 4:07 AM
>To: brewster@archive.org
>Subject: Where did 44 days come from?

>

>

>Dear Mr. Kahle,

>

> I am currently researching the integrity of URLs for my PhD thesis,
>provisionally titled 'Managing Information Flow on the Web', and have
>frequently come across your work in archiving the Internet during my
>research. In particular, the article you published in
>Scientific American
>entitled 'Preserving the Internet' (March 1997,
>www.sciam.com/0397issue/0397kahle.html) appears to be the most widely
>referenced article on the lifetime of URLs, with your figure of 44 days
>being quoted as the accepted standard in many other papers.

>

> As such, I would be grateful if you could let me know from
>which study

>this figure was obtained, as it is not referenced in your
>article. I have

>located a paper from Chankhunthod, Danzig et al, ("A
>hierarchical Internet

>Object Cache",

>[http://catarina.usc.edu/danzig/cache/subsubsectionstar3_4_0_1.html#SECTION00](http://catarina.usc.edu/danzig/cache/subsubsectionstar3_4_0_1.html#SECTION00040100000000000000)

040100000000000000)

which gives a figure of 44 days for object lifetime, but I am reluctant to use it in my thesis with reference to your work, as I cannot find any explicit link between the two, other than the figure '44 days'. Is this the study you were referring to? If not, could you let me know which was?

Thank you, in advance, for your help in this matter. I look forward to your reply.

Yours

Mike Evans

The Network Research Group
University of Plymouth
Plymouth
Devon
UK

APPENDIX D

LIST OF PUBLICATIONS

During the course of this research project, the author has contributed to 12 published papers, as detailed below. Several of the papers relate directly to the focus of the research project, whereas others are associated with further work in which the author was involved during the research period.

1. Evans, M.P., Kettunen, K.T., Blackwell, G.K., Furnell, S.M., Phippen, A.D., Hope S. and Reynolds, P.L. (1997), "Network Resource Adaptation in the DOLMEN Service Machine", In: *Intelligence in Services and Networks: Technology for Cooperative Competition*, Mullery et al. (eds.), Springer, 1997.
2. Evans, M.P., Phippen, A.D., Furnell, S.M. and Reynolds, P.L. (1997), "Resource Adaptation in the TINA Service Environment", *Proceedings of Fourth Communications Networks Symposium*, Manchester, UK, 7-8 July 1997.
3. Liljeberg, M., Evans, M., Furnell, S., Maumon, N., Raatikainen, K., Veldkamp, E., Wind, B. and Trigila, S. (1997), "Using CORBA to Support Terminal Mobility", *Proceedings of TINA 97 Conference*, Santiago, Chile, 17-21 November 1997.

4. Evans, M.P., Furnell, S.M., Phippen, A.D., Reynolds, P.L. (1998), "Mobility Considerations for integrated Telecommunications Service Environments", *Proceedings of IEE Sixth International Conference on Telecommunications*, Edinburgh, UK, 29 March-1 April 1998
5. Evans, M.P., Phippen, A.D., Mueller, G., Furnell, S.M., Sanders, P.W. and Reynolds, P.L. (1998), "Content Migration on the World Wide Web", *Proceedings of the first International Network Conference 1998 (INC '98)*, Plymouth, UK, 6-9 July 1998: 156-161.
6. Evans, M.P. Phippen, A.D., Mueller, G., Furnell, S.M., Sanders, P.W. and Reynolds, P.L. (1999), "Strategies for Content Migration on the World Wide Web", *Internet Research*, vol. 9, no. 1, 1999. pp25-34.
7. Reynolds, P., Furnell, S., Evans, M. and Phippen, A. (1999), "A Hyper Graphics Markup Language for optimising WWW access in wireless networks", *Proceedings of Euromedia 99*, Munich, Germany, 25-28 April 1999: 136-144
8. Furnell, S., Evans, M., Phippen, A., Ali AbuRgheff, M. (1999) "Online Distance Learning: Expectations, Requirements and Barriers", *Virtual University Journal*, vol. 2, no. 2.

9. Furnell, S.M., Evans, M.P. and Dowland, P.S. (2000), "Developing tools to support online distance learning", *Proceedings of EUROMEDIA 2000*, Antwerp, Belgium, 8-10 May 2000.
10. Evans, M.P. and Furnell, S.M. (2000) "Internet-based security incidents and the potential for false alarms", *Internet Research*, vol. 10, no. 3: 238-245. ,2000
11. Furnell, S.M., Evans, M.P. and Bailey, P. (2001), "The promise of Online Distance Learning: Addressing academic and institutional concerns", *Quarterly Review of Distance Education*, vol. 1, no. 4: 281-291
12. Evans, M.P. and Furnell, S.M. (2001), "The Resource Locator Service: Fixing a Flaw in the Web", To appear in *Computer Networks Journal - The International Journal of Computer and Telecommunications Networking*, Elsevier Science

In addition, the author has contributed to a chapter in a book, as detailed below.

Furnell, S.M., Warren, M.J. and Evans, M.P. (2001), "The ISHTAR World Wide Web Dissemination and Advisory Service for Healthcare Information Security", in *Implementing Secure Healthcare Telematics Applications in Europe*. The ISHTAR Consortium (Eds). Technology and Informatics 66, IOS Press: pp249-280.

Finally, the author has also contributed to the World Wide Web standards process through the publication of a World Wide Web Consortium (W3C) NOTE, as detailed below.

Evans, M.P., Furnell, S.M., Phippen, P., Reynolds, P., Lilly, N. and Hammac, J., “Hyper Graphics Markup Language (HGML)”, W3C NOTE, 19th June 1998, <http://www.w3.org/TR/NOTE-HGML>

Copies of the papers most closely related to the research described are bound within this appendix of the thesis.

Content Migration on the World Wide Web

M.P.Evans[†], A.D.Phippen[‡], G.Mueller[†], S.M.Furnell[†], P.W.Sanders[†], P.L.Reynolds[†]

Network Research Group

[†] School of Electronic, Communication and Electrical Engineering, University of Plymouth, Plymouth, UK.

[‡] School of Computing, University of Plymouth, Plymouth, UK.

e-mail contact: Mike.Evans@jack.sec.plym.ac.uk

ABSTRACT

The World Wide Web has experienced explosive growth as a content delivery mechanism, delivering hypertext files and static media content in a standard and consistent way. However, this content has not been able to interact with other content, making the web a distribution system rather than a distributed system. This is beginning to change, however, as distributed component architectures such as CORBA, Enterprise JavaBeans, and DCOM are being adapted to work with the web's architecture. This paper tracks the development of the web as a distributed platform, and highlights the potential to employ an often neglected feature of distributed computing: migration. The paper argues, however, that all content on the web, be it static images or distributed components, should be free to migrate according to either the policy of the server, or the content itself. The paper goes on to describe the requirements of such a content migration mechanism, and shows how, combined with network traffic profiling, a network can be optimised by dynamically migrating content according to traffic demand.

1. Introduction

1.1 Software Resources

The World Wide Web ('the web') is a platform for distributing software resources across the Internet, which are then presented as rich, consistent content by applications on the client (usually a browser). The three main standards which define the platform are:

- the Uniform Resource Locator (Berners-Lee, et al 1994);
- HyperText Transfer Protocol (Berners-Lee, et al, 1996);
- HyperText Markup Language (Berners-Lee, et al, 1995).

The Uniform Resource Locator (URL) is used to locate software resources; the HyperText Transfer Protocol (HTTP) is the protocol used to distribute the resources; and HyperText Markup Language (HTML) is used to present the information contained within the software resources in a consistent way across all computer platforms.

As such, today's web is a large distribution system. The software resource is a single, self-contained unit of data (usually a binary or text file), which the web can locate (using the URL) and distribute (using HTTP). It encodes content, which is presented on the client by applications according to the media type the content represents (e.g. images, video, etc.). Each media type must conform to its own universal standard, which is not part of the specification of the web itself, but which contributes to its ubiquity and openness. The content is decoded from the software resource by the browser or its own application (generally termed a 'viewer' or 'plug-in'), and is presented consistently across all platforms according to the layout and style specified by the HTML page. For example, the GIF (Graphics Interchange Format) standard, developed by CompuServe, is a standard format for

compressing and encoding images. A GIF viewer is an application which works inline with the browser to interpret a GIF image file and display the image it contains. This GIF viewer essentially reads in a generic, platform-independent file (the software resource) which contains an encoding of the image, and converts the encoded data into content: platform-dependent information which can be displayed on the client's screen as the decoded image in a consistent way across all platforms according to the layout and style specified by the HTML. The same can also be said for other content formats (e.g. JPEG, MPEG, AVI, QuickTime), each of which encodes a specific media type according to the media's own defined standard. In fact, for any type of content to proliferate on the web, it must have its own platform-independent standard with its own platform-specific viewers generally available on every platform. To the web's distribution mechanism (i.e. the web servers and HTTP), everything is a generic software resource (see Figure 1). Only when the correct application receives it on the client does it become content.

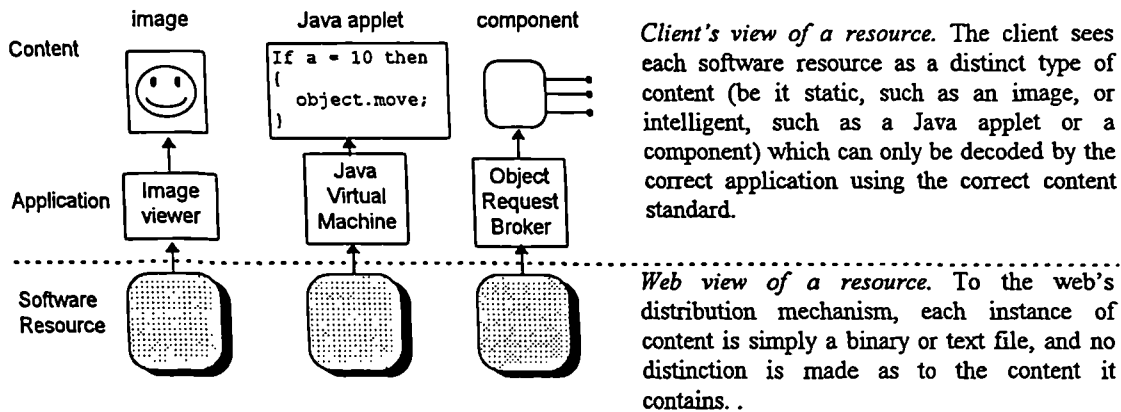


Figure 1: Relationship between content and the software resource

1.2 Static and Intelligent Content

This content has traditionally consisted of static files without functionality, and without the ability to interact with other software resources. A GIF file, for example, contains the information required to display the image it encodes with a suitable viewer, but there is no computational intelligence contained within it; consequently, the user cannot interact with it (the use of 'image maps' within a browser, whereby a user can click on an image to navigate to another page, is controlled and formatted by the HTML in the web page, not the image file). Currently, then, the web is a *distribution* system, not a distributed system. However, this is changing. As the web matures, its functionality is increasing, and, more importantly, the intelligence contained within the resources it is currently distributing is growing along with the web itself. To distinguish between resources which contain some form of static media content (such as an image), and resources which have some form of computational intelligence as part of their content (such as a Java applet), this paper will define the terms *static content* and *intelligent content*, respectively.

Intelligent content currently consists of small self-contained blocks of code which reside on a server as software resources, and are downloaded onto a client machine, where they are executed by a suitable application, usually inline with an HTML page. Java applets are an example of such content, as are Microsoft's ActiveX controls. This type of content is limited, however, by its self-contained nature: a Java applet, for example, cannot communicate with other Java applets on machines other than the server it originated from. In order to distribute the intelligence of a large scale application, the components of the application must be able to interact with each other across a distributed environment; to achieve this, a distributed component architecture must be employed.

2. Distributed Components

Component software develops on the potential of object-based software by constructing software from components which encapsulate functionality and data. This is similar to object orientation, but allows dynamic component interaction at runtime (known as 'runtime reuse'). This is achieved through the use of a *component architecture*, which is a defined platform with rules for interaction between components. Any component developed on top of this platform will be able to interact with any other component built on the same platform. Whilst a general component architecture enables components on the same computer to interact, distributed component architectures add to the functionality by enabling interaction across a distributed network environment. A client component can not only use the services of components on its host machine, but also any other machine which supports the distributed architecture. Components within such architectures are also termed distributed objects, primarily because the architecture itself is based on the object-oriented paradigm. Currently, the distributed component field is dominated by two major architectures: Microsoft's Distributed Component Object Model (DCOM) and the Object Management Group's Common Object Request Broker Architecture (CORBA).

DCOM is the distributed extension to Microsoft's COM (Component Object Model), and extends the basic functionality to incorporate transparent network distribution and network security mechanisms into the architecture. Through DCOM, ActiveX controls can interact with one another, and with other COM-based components, across a network.

CORBA is a complete architectural specification developed by the Object Management Group (OMG, 1995) which specifies both a component architecture, and component services. CORBA is entirely generic, defining platform-independent data-types, protocols for communication across platforms, and a number of platform-independent services which provide the components with a number of useful services such as security, transaction processing, and naming and location services for finding components across a distributed system. CORBA's functionality is implemented through an Object Request Broker (ORB), which provides the transparencies required by the architecture.

Both architectures offer the developer similar features and similar benefits. They both provide a component distribution mechanism employing network and location transparency, marshalling mechanisms, etc., and both expose functionality through language-independent interfaces. They are reliable distributed platforms upon which large scale distributed applications can be built.

2.1 Distributed Components and the WWW

Such distributed component systems are increasingly being incorporated into the web. Distributed components are becoming the next type of software resource to share server space with existing types of static and intelligent content. This allows the web to become a true distributed system, being able to provide distributed applications and services via a client's browser. Netscape, for example, has integrated CORBA functionality into its Communicator 4.0 browser, allowing it to interact with CORBA components on CORBA-enabled servers. Equally, Microsoft's Internet Explorer 4.0 browser is DCOM-enabled, allowing it to communicate with DCOM components on DCOM-enabled servers. In this way, the web is evolving into a complete distributed system, termed the 'Object Web' (Orfali et al, 1996) to reflect the object-based nature of the distributed architectures being employed.

3. Content Migration

3.1 An Overview of Migration in a Distributed System

One of the benefits of a distributed system is the ability of an application to be distributed across multiple hosts across a network, such that no one host has to execute the whole application on its own. With a fast enough network, this 'load balancing' functionality can greatly increase the efficiency and performance of the application in a way which is entirely transparent to the client machine. However, the drawback to this distributed paradigm is the static nature of the location of each component. Once a component has been installed on a host, it cannot easily be moved to another host. Thus, should the host's, or its network's, performance degrade in any way, access to the component will be affected. Invocations on the component's interfaces will be slowed down, which in turn will impact on the performance of the application as a whole. The component can be manually relocated to a different host, but this is time-consuming. Most distributed applications comprise many components, and it would be impractical to manually redistribute them all whenever necessary.

As such, various automatic component relocation mechanisms exist. These 'migration mechanisms' can transparently move a component from one host to another such that the client has no awareness of the move. These mechanisms are provided by some (though not all) distributed architectures as a way of dynamically relocating components to provide load balancing and fault tolerance. Distributed component architectures can migrate entire components, including their functionality and data, and retain the state of the component from one machine to another. Such migration mechanisms can optimise a network, and can enable mobile agents: autonomous objects of code which are free to roam across a network.

3.2 The Problems with Distributed Components and the WWW

The distributed component is a new type of intelligent content, which has the ability to interact with other content of the same type. However, components of different architectures cannot directly communicate with each other. Thus, Netscape's CORBA-compliant browser cannot use DCOM components, and Microsoft's DCOM-enabled browser cannot use CORBA components. As such, neither component architecture provides its content (the distributed component) with true ubiquity across the web in a way in which traditional content does.

This problem impacts on the architectures' use of migration. Most, including DCOM and Enterprise JavaBeans, do not support migration at all. However, even if they did, current distributed architectures cannot successfully employ a ubiquitous migration mechanism across the web, because no matter how open they are, the type of resource which can be migrated is tied too closely to the architecture itself. The web treats each software resource as a generic unit. The URL is used to reference it, and HTTP to distribute it, regardless of the resource's content type. In contrast, distributed architectures work only with their own content, and use their own reference formats to locate the components. Thus, only components created specifically to an architecture's specifications can be migrated, and only if both hosts involved in the migration support the architecture. Currently, however, the vast majority of content on the web today consists of JPEG and GIF files, and Java applets, which have no concept of a distributed architecture, much less the services that one can provide. Equally, servers supporting CORBA or DCOM are in the minority meaning there are physically very few places for a component to migrate to.

3.3 Requirements for a Migration Mechanism on the WWW

For a migration mechanism to be successful on the web, it must recognise the diverse range of content that exists. As such, it must be completely decoupled from the content that it can migrate, and instead focus on the software resource: a generic unit of data which may or may not be aware of the mechanism (see Figure 2). Additionally, to truly be of benefit, the mechanism must fit in with the existing web architecture. As such, it must reference each resource using a standard URL, not an architecture-specific reference. With so many businesses using URLs on their promotional material, any mechanism which required a new format for resource location would not be accepted.

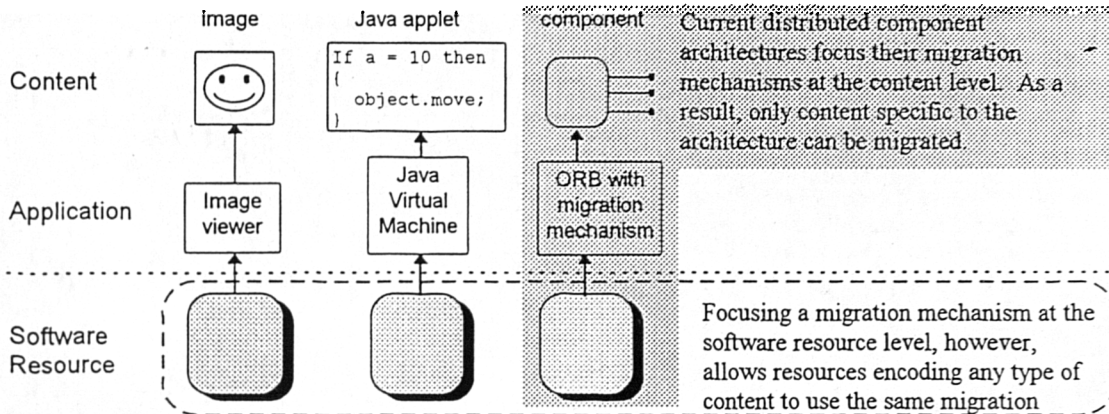


Figure 2: Migration Mechanisms at the Content and Resource Levels

Within a distributed system, much use is made of the term ‘transparency’ (RM-ODP, 1995). This is used to convey the concept that the services performed by the distributed system (such as migration) happen without components being aware that anything has changed. Thus, a transparent migration mechanism is one in which components are migrated to another machine without the component, or a client wishing to access the component, being aware of the move. However, such a mechanism can be made ‘translucent’; that is, the components can be moved transparently, but if they require the service themselves, they can use it to initiate their own migration. In this way, the migration is controlled by the component rather than the server hosting the software resource. For example, static content has no intelligence, and so cannot make use of a migration mechanism. As such, if the resource encoding the static content is to be migrated, it must be at the server’s discretion. The server is therefore able to migrate the resource without the resource or any other host being aware of the move. Intelligent content, however, has the ability to use any service the network can provide. Thus, a migration mechanism can be used by intelligent content to migrate itself, transforming it into a mobile agent.

In this way, a translucent migration mechanism on the world wide web can provide a host of new and extended services. The same mechanism can be used by intelligent content (to autonomously roam the web), and by web servers, (to optimise the network); it can solve the ‘broken link’ problem typical of hypertext documents, whereby a URL embedded within an HTML document is rendered useless when the content it refers to is moved. It can also be employed on a company’s intranet, allowing resources to migrate freely, either of their own volition, or transparently by the server hosting them. By providing its servers with the ability to monitor their own performance, a company can simply connect a new server to its intranet, and wait for resources to migrate to it from existing servers under strain. Using dynamic network configuration protocols, and wireless network technology such as Wireless LAN, this facility can be extended such that a server need only be brought into range of a mobile basestation, and switched on: the server will connect to the network, and the resources will populate the server, automatically.

3.3.1 Identifying the Content to Migrate through Network Traffic Profiling

Profiling network traffic can identify which content should migrate and when, enabling a network to be optimised and managed more effectively. Currently, certain network technologies and Service Level Agreements (SLAs) with network providers insist on the network user specifying the expected quality of service of the network at certain times of the day. For example, Frame Relay can ensure a certain throughput to the user over a short period of time by guaranteeing a Committed Information Rate (CIR). This CIR is the rate which is, on average, available to the user.

Determining the CIR is a difficult process and involves a good knowledge of the local traffic. The network manager has to plan for the expected traffic, keeping in mind that at very busy times he does not have the same throughput and availability of capacity above the CIR for bursty traffic. Depending on the use of protocols, this can have a major influence on the choice of the CIR. Frequently this choice is not influenced by the traffic volume itself, but by the budget a company is able to afford. However, choosing too low a CIR can cause congestion, data loss, poor throughput, and a financial loss. Traffic profiling is very important in such networks, whereby the traffic is monitored in order to determine the quality of service required. Research by the members of the author team is developing a methodology for profiling traffic in this way. It has been determined that whilst overall network traffic may be variable over the short-term, over time it only increases. The SLAs, therefore, can provide the business case for introducing content migration as a means of balancing the network and staying within the CIR. With a transparent migration mechanism built into a company's intranet, software resources can be migrated to balance the load not just across servers, but across the network. A traffic profiling system can be used to monitor the traffic on a company's network. If network traffic has increased at a particular point, resources can be migrated to ease the flow of traffic at the bottleneck. If the traffic is too great only at certain times of day, the profile will show this, and the resources can be migrated back and forth according to the time of day.

4. Conclusion

A transparent, content-independent migration mechanism for the web, combined with existing distributed component architectures and sophisticated network traffic profiling techniques should optimise both a server and the network, and can provide a new class of services to users. Such a mechanism is currently being developed by the authors, and an initial design has been produced which can provide such migration services without breaking the existing Internet architecture. That is, the URL is retained for locating resources, the resources can be of any content type, the mechanism can be integrated gradually into the existing infrastructure, and the mechanism imposes no foot-print on the client. This mechanism is currently work in progress and will be expanded upon in future publications.

5. References

Berners-Lee, T.; Masinter, L. and McCahill, M. (1994) Uniform Resource Locators (URL), *RFC-1738*, CERN/Xerox/University of Minnesota, December 1994

Berners-Lee, T., Connolly, D. (1995) HyperText Markup Language - 2.0, *RFC 1866*, MIT/W3C, November 1995

Berners-Lee, T., Fielding, R. and Frystyk, H. (1996) Hypertext Transfer Protocol - HTTP/1.0, *RFC 1945*, MIT/UCS/UC Irvine, May 1996.

OMG (1995) "The Common Object Request Broker: Architecture and Specification, Revision 2.0", Object Management Group, 1995

Orfali, R., Harkey, D., Edwards, J.(1996) *The Essential Client/Server Survival Guide*, 1996, Wiley

RM-ODP (1993) "Open Distributed Processing Reference Model (RM-ODP)", ISO/IEC DIS 10746-1 to 10746-4, 1995. <http://www.iso.ch:8000/RM-ODP/>

Strategies for content migration on the World Wide Web

M.P. Evans
A.D. Phippen
G. Mueller
S.M. Furnell
P.W. Sanders and
P.L. Reynolds

The authors

M.P. Evans, G. Mueller, S.M. Furnell, P.W. Sanders and P.L. Reynolds are all at the Network Research Group, School of Electronic, Communication and Electrical Engineering, University of Plymouth, Plymouth, UK.

A.D. Phippen is at the School of Computing, University of Plymouth, Plymouth, UK.

E-mail: Mike/Evans@jack.see.plym.ac.uk

Keywords

Distributed data processing, Distribution, Internet

Abstract

The World Wide Web has experienced explosive growth as a content delivery mechanism, delivering hypertext files and static media content in a standardised way. However, this content has been unable to interact with other content, making the Web a distribution system rather than a distributed system. This is changing, however, as distributed component architectures are being adapted to work with the Web's architecture. This paper tracks the development of the Web as a distributed platform, and highlights the potential to employ an often neglected feature of distributed computing: migration. Argues that all content on the Web, be it static images or distributed components, should be free to migrate according to either the policy of the server, or the content itself. The requirements of such a content migration mechanism are described, and an overview of a new migration mechanism, currently being developed by the authors, is presented.

Introduction

The World Wide Web (the Web) is a platform for distributing software resources across the Internet, which are then presented as rich, consistent content by applications on the client (usually a browser). The three main standards which define the platform are:

- (1) the Uniform Resource Locator (Berners-Lee *et al.*, 1994);
- (2) HyperText Transfer Protocol (Berners-Lee *et al.*, 1996);
- (3) HyperText Markup Language (Berners-Lee *et al.*, 1995).

The Uniform Resource Locator (URL) is used to locate software resources; the HyperText Transfer Protocol (HTTP) is the protocol used to distribute the resources; and HyperText Markup Language (HTML) is used to present the information contained within the software resources in a consistent way across all computer platforms.

Consequently, today's Web is a large distribution system. The software resource is a single, self-contained unit of data (usually a binary or text file), which the Web can locate (using the URL) and distribute (using HTTP). It encodes content, which is presented on the client by applications according to the media type the content represents (e.g. images, video, etc.). Each media type must conform to its own universal standard, which is not part of the specification of the Web itself, but which contributes to its ubiquity and openness. The content is decoded from the software resource by the browser or its own application (generally termed a "viewer" or "plug-in"), and is presented consistently across all platforms according to the layout and style specified by the HTML page. For example, the graphics interchange format (GIF) standard, developed by CompuServe, is a standard format for compressing and encoding images. A GIF viewer is an application which works inline with the browser to interpret a GIF image file and display the image it contains. This GIF viewer essentially reads in a generic, platform-independent file (the software resource) which contains an encoding of the image, and converts the encoded data into content: platform-dependent information which can be displayed on the client's screen as the decoded image in a consistent way across all

platforms according to the layout and style specified by the HTML. The same can also be said for other content formats (e.g. JPEG, MPEG, AVI, QuickTime), each of which encodes a specific media type according to the media's own defined standard. In fact, for any type of content to proliferate on the Web, it must have its own platform-independent standard with its own platform-specific viewers generally available on every platform. To the Web's distribution mechanism (i.e. the Web servers and HTTP), everything is a generic software resource (see Figure 1). Only when the correct application receives it on the client does it become content.

Static and intelligent content

Web content has traditionally consisted of static files without functionality, and without the ability to interact with other software resources. A GIF file, for example, contains the information required to display the image it encodes with a suitable viewer, but there is no computational intelligence contained within it; consequently, the user cannot interact with it (the use of "image maps" within a browser, whereby a user can click on an image to navigate to another page, is controlled and formatted by the HTML in the Web page, not the image file). Currently, then, the Web is a distribution system, not a distributed system. However, this is changing. As the Web matures, its functionality is increasing, and, more important, the intelligence contained within the resources it is currently distributing is growing along with the Web itself. To distinguish between resources which contain some form of static media

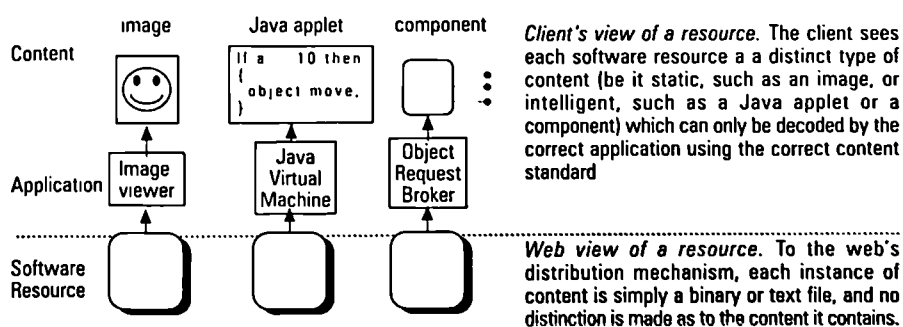
content (such as an image), and resources which have some form of computational intelligence as part of their content (such as a Java applet), this paper will define the terms static content and intelligent content, respectively.

Intelligent content currently consists of small self-contained blocks of code which reside on a server as software resources, and are downloaded onto a client machine, where they are executed by a suitable application, usually inline with an HTML page. Java applets are an example of such content, as are Microsoft's ActiveX controls. This type of content is limited, however, by its self-contained nature: a Java applet, for example, cannot communicate with other Java applets on machines other than the server it originated from. In order to distribute the intelligence of a large scale application, the components of the application must be able to interact with each other across a distributed environment; to achieve this, a distributed component architecture must be employed.

Distributed components

Component software develops on the potential of object-based software by constructing software from components which encapsulate functionality and data. This is similar to object orientation, but allows dynamic component interaction at runtime (known as "runtime reuse"). This is achieved through the use of a component architecture, which is a defined platform with rules for interaction between components. Any component developed on top of this platform will be able to interact with any other component built on the same platform. While a general component architecture enables

Figure 1 Relationship between content and the software resource



components on the same computer to interact, distributed component architectures add to the functionality by enabling interaction across a distributed network environment. A client component can use the services of components not only on its host machine, but also any other machine which supports the distributed architecture. Components within such architectures are also termed distributed objects, primarily because the architecture itself is based on the object-oriented paradigm. Currently, the distributed component field is dominated by two major architectures:

- (1) Microsoft's distributed component object model (DCOM); and
- (2) the Object Management Group's common object request broker architecture (CORBA).

DCOM is the distributed extension to Microsoft's component object model (COM), and extends the basic functionality to incorporate transparent network distribution and network security mechanisms into the architecture. Through DCOM, ActiveX controls can interact with one another, and with other COM-based components, across a network.

CORBA is a complete architectural specification developed by the Object Management Group (OMG, 1995) which specifies both a component architecture, and component services. CORBA is entirely generic, defining platform-independent data-types, protocols for communication across platforms, and a number of platform-independent services which provide the components with a number of useful services such as security, transaction processing, and naming and location services for finding components across a distributed system.

CORBA's functionality is implemented through an object request broker (ORB), which provides the transparencies required by the architecture.

Both architectures offer the developer similar features and similar benefits. They both provide a component distribution mechanism employing network and location transparency, marshalling mechanisms, etc., and both expose functionality through language-independent interfaces. They are reliable distributed platforms on which large scale distributed applications can be built. Such distributed component systems are increasingly being incorporated into

the Web. Distributed components are becoming the next type of software resource to share server space with existing types of static and intelligent content. This allows the Web to become a true distributed system, being able to provide distributed applications and services via a client's browser. Netscape, for example, has integrated CORBA functionality into its Communicator 4.0 browser, allowing it to interact with CORBA components on CORBA-enabled servers. Equally, Microsoft's Internet Explorer 4.0 browser is DCOM-enabled, allowing it to communicate with DCOM components on DCOM-enabled servers. In this way, the Web is evolving into a complete distributed system, termed the "object Web" (Orfali *et al.*, 1996) to reflect the object-based nature of the distributed architectures being employed.

Content migration

An overview of migration in a distributed system

One of the benefits of a distributed system is the ability of an application to be distributed across multiple hosts across a network, in such a way that no one host has to execute the whole application on its own. With a fast enough network, this "load balancing" functionality can greatly increase the efficiency and performance of the application in a way which is entirely transparent to the client machine. However, the drawback to this distributed paradigm is the static nature of the location of each component. Once a component has been installed on a host, it cannot easily be moved to another host. Thus, should the host's, or its network's, performance degrade in any way, access to the component will be affected. Invocations on the component's interfaces will be slowed down, which in turn will affect the performance of the application as a whole. The component can be manually relocated to a different host, but this is time-consuming. Most distributed applications comprise many components, and it would be impractical to manually redistribute them all whenever necessary.

Consequently, various automatic component relocation mechanisms exist. These "migration mechanisms" can transparently move a component from one host to another in such a way that the client has no awareness of the move. These

mechanisms are provided by some (though not all) distributed architectures as a way of dynamically relocating components to provide load balancing and fault tolerance. Distributed component architectures can migrate entire components, including their functionality and data, and retain the state of the component from one machine to another.

The problems with distributed components and the WWW

The distributed component is a new type of intelligent content, which has the ability to interact with other content of the same type. However, components of different architectures cannot directly communicate with each other. Thus, Netscape's CORBA-compliant browser cannot use DCOM components, and Microsoft's DCOM-enabled browser cannot use CORBA components. Thus, neither component architecture provides its content (the distributed component) with true ubiquity across the Web in the way in which traditional content does.

This problem affects the architectures' use of migration. Most, including DCOM and Enterprise JavaBeans, do not support migration at all. However, even if they did, current distributed architectures cannot successfully employ a ubiquitous migration mechanism across the Web, because no matter how open they are, the type of resource that can be migrated is tied too closely to the architecture itself. The Web treats each software resource as a generic unit. The URL is used to reference it, and HTTP to distribute it, regardless of the resource's content type. In contrast, distributed architectures work only with their own content, and use their own reference formats to locate the components. Thus, only components created specifically to an architecture's specifications can be migrated, and only if both hosts involved in the migration support the architecture. Currently, however, the vast majority of content on the Web today consists of JPEG and GIF files, and Java applets, which have no concept of a distributed architecture, much less the services that one can provide. Equally, servers supporting CORBA or DCOM are uncommon, leaving very few places for a component to physically migrate to.

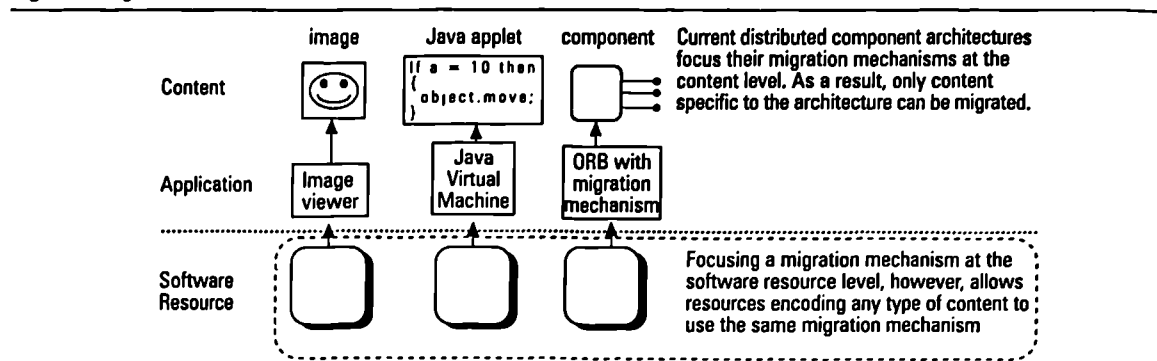
Requirements for a migration mechanism on the WWW

For a migration mechanism to be successful on the Web, then, it must recognise the diverse range of content that exists. Therefore, it must be completely decoupled from the content that it can migrate, and instead focus on the software resource: a generic unit of data which may or may not be aware of the mechanism (see Figure 2). Additionally, to truly be of benefit, the mechanism must fit in with the existing Web architecture, rather than build its own set of standards on top of the existing Web platform. In this way, it can be used by existing Web content as much as by intelligent content such as distributed components, and can provide services which distributed components can use to enable the Web to become a distributed platform.

True content migration, then, where content of any type can be freely migrated, relies on implementing migration at the resource level. In order to achieve this, the following set of requirements for a resource-level migration mechanism have been identified:

- *Universal client access.* The mechanism must be accessible to clients of any type and should *not require clients to be altered in order to use it*. Thus, existing software does not need to be rewritten, and future software will not require any extra facilities in order to use it.
- *Content neutrality.* A Web-based mechanism must be completely decoupled from the content it can migrate, enabling it to migrate all resources, no matter what type of content they encapsulate (see Figure 2).
- *Full integration with the Web's current architecture.* The mechanism must reuse as much of the Web's existing architecture as possible. Specifically, this means the reuse of HTTP and the URL. There is too much investment in the infrastructure supporting HTTP to change it overnight, and the URL is becoming accepted by the public as the only way to navigate to Web resources. With businesses now using the URL as part of their advertising campaigns, URLs can now be recognised even by people without access to the Web.
- *Practical design.* Resource migration can be technically achieved in many different ways, but adopting a practical approach means focusing on the requirements of Web

Figure 2 Migration mechanisms at the content and resource levels



developers, existing Web software, and (most important) Web users, rather than focusing on a technically optimal design. A practical design also means one that takes into account the dynamics and characteristics of the Web (and, by implication, the users of the Web); an approach that technically works will not achieve ubiquity if it results in the Web appearing to run more slowly.

In the next section, this set of requirements will be used to evaluate existing approaches to migration to see which is best suited to the development of a migration mechanism for the Web.

Developing a Web-based migration mechanism

Methods of resource migration

For any migration mechanism, there are four different methods through which a resource can be tracked once migration has occurred (Ingham *et al.*, 1996). These are:

- (1) Forward referencing.
- (2) Name service.
- (3) Callback.
- (4) Search.

Forward referencing

Forward referencing involves leaving behind a reference in place of the migrated resource which points to the resource's new location. Thus, an object leaves behind a chain of references on each host it visits. For example, the migration mechanism of the "W3Objects" system (Ingham *et al.*, 1996), and "Voyager", from ObjectSpace (an agent-oriented CORBA implementation), both adopt this approach.

When an object migrates in Voyager, a "virtual reference" is left behind to forward messages to the new location. As an object migrates, more virtual references are created, forming long chains which eventually resolve onto the object itself. The W3Objects approach is similar, in that "forward references" are created each time a resource migrates; however, to prevent long chains building up, "shortcuts" can be created which allow a reference holder (that is, a resource with a link to the migrated resource) to bypass the chain of references, and reference the resource directly.

Suitability for the Web

Voyager's mechanism is unsuitable for the Web as, like most other distributed architectures, it only migrates Voyager-aware content, and is therefore not content-neutral. Surprisingly, the mechanism used by W3Objects will also only work with its own, object-oriented resource (termed a "W3Object") and a specially-defined reference (termed a "W3reference"), and so it too is not content-neutral. Furthermore, in order to use the W3Objects system, each client's browser must be adapted to work with W3References rather than URLs.

However, the forward reference method itself is unsuitable for use on the Web. Each link in the chain of forward references adds another point of potential failure (Ingham *et al.*, 1996), and if the chain breaks, then the resource is lost completely. Further, the characteristics of the Web will make managing the chains unrealistic, as the number of forward references will increase with both time (some resources, such as autonomous agents, will migrate constantly) and space (every resource will require a chain of references to be maintained).

Name service

The name service method employs an external system to maintain references to registered resources at all times. Such mechanisms generally focus on the use of the name used to identify a resource, and attempt to abstract any location-dependent information out of the name itself. For example, the uniform resource name (URN) is a proposed standard by the Internet Engineering Task Force (IETF) for naming a resource independently from its location (Sollins and Masinter, 1994). Specifically, a URL is used to locate a resource, while a URN can be used to identify a resource (Berners-Lee *et al.*, 1994). The URN can then be mapped onto the URL through an external resolver discovery service (RDS), which maintains the location of the resource. Should the resource have migrated, the RDS will resolve the URN into a URL that points to another RDS which can resolve the URL. Thus, a chain of references is built up across the resolver service, rather than across each visited server.

Suitability for the Web

The URN identifies a resource independently from its location, and so subsumes the URL, treating it not as a name, but as a pointer to a location. Thus, while the URN has content-neutrality, it does not support full integration, as the URL cannot be used at the user level.

Also the name service method suffers from the same problems inherent with any “chain” of references, as described above. Further, the method is not practical, as it does not take into account the characteristics of the Web users: it requires, for example, that a resource’s name remain invariant throughout its lifetime (which can be “for hundreds of years” (Sollins and Masinter, 1994)), but in real life, the ownership of a resource can change within its lifetime, and the new owner may wish to give the resource a new name.

Callback

The callback method relies on a resource to inform all other resources with references to it of any change in its location, in order to ensure referential integrity. The benefit of this approach is that there is no indirection, and so no chain of references need be maintained.

The Hyper-G system (Kappe, 1995) adopts this approach, maintaining a large database on

the references used between resources. Should a resource move, the database is informed, and all references are updated. This is similar to the name service approach, in that an external service is used, but it is the relationships between resources which are maintained by the service, rather than the resources’ locations.

Suitability for the Web

This approach either requires each resource to know which other resource has references to it, or requires an external service to maintain the references. However, the former approach is unrealistic, as the Web is a federated system, with no central control: a resource has no way of knowing who or what is referencing it. Equally, the latter approach is unrealistic, as the size of the database of references would become impossible to manage, and many Web servers are frequently offline, resulting in the database being swamped as it must store pending reference updates until they are online again (Briscoe, 1997).

Search

The search method does not attempt to update the references between resources, or to maintain the location of a resource. Rather, it uses a sophisticated search mechanism to find the resource if it migrates. To ensure success, the entire system must potentially be searched, which involves flooding the network. This has the advantage that so long as the server hosting a particular resource is accessible, the resource can be guaranteed to be found, as the flood will eventually cover all servers. Thus, the search approach has perfect robustness. The Harvest information system (Mic Bowman *et al.*, 1995) uses this approach to catalogue and index a distributed system’s collection of resources. However, the Harvest system is used to index and search for pertinent information within resources, and so is effectively a search engine which can index an entire distributed system.

Suitability for the Web

While flooding a network provides perfect robustness, it is also the most costly method in terms of messaging overheads (Ingham *et al.*, 1996). A flooding algorithm must be implemented which spans the entire Web. To prevent the network being overwhelmed with packets (which, unchecked, would increase

exponentially), attempts must be made to restrict the flood. This can be achieved by including time to live fields in any messages sent by such a mechanism, but this requires knowledge of the exact diameter of the Web (Tanenbaum, 1996).

Selecting a migration method

The callback approach

The callback service can be immediately ruled out. As has been said, a resource on the Web has no way of knowing who or what is referencing it, and so any implemented callback service simply cannot be used.

The chain approach

The forward reference and the name service approaches can be grouped together and termed the “chain approach”, as both rely on a chain references to effect migration. The difference is simply that the forward reference approach leaves its references on the servers it has visited, while the name service approach relies on a separate service to store and maintain its chain of references. The concept of the chain approach, then, can be examined in its own right, but does not meet all of the requirements specified above. The very fact that a chain exists exposes the whole approach to the chain’s weakest link; in this case, the weakest link is the most unreliable server within the chain, meaning that a resource may be lost because somebody else’s server has crashed. Finding that server can be difficult; worse, the resource’s owner will have no control over the maintenance of the crashed server, and if it goes down permanently, the resource may be lost permanently. This is not just impractical, it is unacceptable to a network such as the Web which is forming the platform for e-commerce: losing a resource can sever the relationship between an organisation and its customers.

The search approach

The search approach comes closest to meeting all of the requirements specified above. Because the search would be performed within the network, the client need not be aware that a search is being performed; it simply receives the resource once it is located. Thus, universal client access is achieved. The search process would be performed using the resource’s URL; consequently, as long as the resource has a

URL, it can be located, regardless of its content type. This achieves the requirement of content neutrality. HTTP and the URL can remain. In fact, so long as the identifier is unique, it can be of any format, leaving the way open for future formats of identifier to be used with the same migration mechanism. Full integration with the Web’s current architecture is, therefore, achieved. However, the message overhead used to locate a resource cannot be ignored. Because it uses a flooding algorithm, the messages will grow exponentially with the size of the network. This is, at best, impractical when considering a network the size of the Internet. Thus, the search approach fails the practical design requirement. If this can be resolved, however, the concept of the search approach is far more robust and scalable than the chain approach. With no chains of references to maintain, and the ability to visit all hosts in a network, there is no weak link in the system. Resources, by definition, cannot be lost. Therefore, adopting a different search algorithm for the search approach could result in a practical search-based migration mechanism on the Web.

Adapting the search approach

The problems described thus far relate to a search algorithm which is parallel in nature, generating exponential traffic as the search progresses, and works on unstructured data. Such an approach cannot be practical on the Web, because its latency overhead occurs at the wrong stage of the migration process. The process of migration can be divided into two stages: first, a resource migrates; then, it must be located whenever a client wishes to use the resource. Generally, the migration stage can cope with higher latency times than the location stage. This is because there is no user interaction with the resource during the migration stage, whereas a resource usually needs to be located because a user wishes to download it. Currently, there is no migration mechanism on the Web; locating a resource is simply a matter of connecting with the appropriate server. Any mechanism that is required to locate a resource will incur its own overhead, and this adds to the latency involved in actually accessing the resource. To the user, this latency is perceived as a slower response time of the Web. With the chain approach, the main overhead occurs

during the migration process. Location is simply a matter of following a chain of references, and so long as this chain is not too large, latency should not be appreciably increased. However, with the parallel search approach described above, all of the overhead occurs during the location process, with the latency increasing as the search continues. Worse, the message overhead also increases (exponentially) as the search continues, resulting in a network with more location traffic than resource traffic.

This, however, is simply one end of a spectrum of search algorithms. For example, another approach could involve constructing a look-up table, with the set of all URLs on the Web being mapped to each respective resource's actual location. The URLs can be ordered as appropriate, and a trivial search algorithm used to locate a specific URL within the look-up table. While this centralised approach is not fault tolerant, and could result in all resources being lost, it does illustrate how structuring the data can fundamentally change the performance of the search approach. What is required, therefore, is an approach which structures the data, but across a distributed system of migration-specific machines.

An overview of a Web-based migration mechanism

This is the approach that is currently being investigated by the authors. A migration mechanism is being developed which uses an external (distributed) service to keep track of the URLs and the actual location of the respective resources. This is similar to the resolver discovery service adopted by the URN approach, and provides the indirection required to retain the format of the URL while allowing the resource to reside on a machine with a different name. However, while the resolver discovery service uses a chain of references to keep track of the migrating resources, the new approach uses what is, essentially, a migration-specific distributed "database". This database is constructed and queried using Web-based technologies, such as Extensible Markup Language (XML). Rather than searching all of the resources on all of the servers across the Web, the set of all resources are represented within this distributed database by their URLs, and it is this database which is searched to locate a resource. Fault

tolerance techniques will be used to ensure no resources are lost, and load balancing will minimise the latency incurred. Because the database contains URLs, any content which can be addressed using a URL can safely migrate using this system. All that is required is for the system to be notified when a migration has occurred. This can be done by the server the resource has migrated from, or the server the resource has migrated to (or, for that matter, by the resource itself, if it contains intelligent content).

Development of this system is currently a work-in-progress, and results from the completed system will be published in a later paper. The next section discusses some of the new services such a system can provide to the Web.

Providing new Web-based services

How a migration mechanism can enable new services

Within a distributed system, much use is made of the term "transparency" (RM-ODP, 1995). This is used to convey the concept that the services performed by the distributed system (such as migration) happen without components being aware that anything has changed. Thus, a transparent migration mechanism is one in which components are migrated to another machine without the component, or a client wishing to access the component, being aware of the move. However, such a mechanism can be made "translucent"; that is, the components can be moved transparently, but if they require the service themselves, they can use it to initiate their own migration. In this way, the migration is controlled by the component rather than the server hosting the software resource. For example, static content has no intelligence, and so cannot make use of a migration mechanism. Therefore, if the resource encoding the static content is to be migrated, it must be at the server's discretion. The server is therefore able to migrate the resource without the resource or any other host being aware of the move. Intelligent content, however, has the ability to use any service the network can provide. Thus, a migration mechanism can be used by intelligent content to migrate itself. It may choose to do this for the purpose of network optimisation (for example, if it detects that the server's performance has degraded due to

excess demand), or it may do this to achieve a goal on behalf of a user. This would effectively enable the intelligent content to become a mobile autonomous agent (Franklin and Graesser, 1996); that is, software which can roam across a network, performing tasks on behalf of a user.

In this way, a translucent migration mechanism on the Web can provide a host of new and extended services. The same mechanism can be used by intelligent content (to autonomously roam the Web), and by Web servers (to optimise the network); it can solve the “broken link” problem typical of hypertext documents, whereby a URL embedded within an HTML document is rendered useless when the content it refers to is moved. It can also be employed on a company’s intranet, allowing resources to migrate freely, either of their own volition, or transparently by the server hosting them. By providing its servers with the ability to monitor their own performance, a company can simply connect a new server to its intranet, and wait for resources to migrate to it from existing servers under strain. Using dynamic network configuration protocols, and wireless network technology such as wireless LAN, this facility can be extended so that a server need only be brought into range of a mobile basestation, and switched on: the server will connect to the network, and the resources will populate the server, automatically.

Mobile servers

Basing the migration mechanism on a search approach effectively provides a service which resolves the IP address of a machine given a specific resource. Thus, the same host can have many different IP addresses over time (for example, if the host is roaming) yet its resources will still be locatable (providing the host is accessible to the migration mechanism), because the mechanism ensures the resource can be located regardless of the current IP address associated with it. This implies that mobile servers can be developed with IP addresses which change according to the server’s location, without affecting the accessibility of the resources being hosted.

Services for distributed component systems

Distributed component systems can use the mechanism to migrate components. Any type of content can use such a migration mechanism,

and this includes intelligent content such as distributed components. Thus the mechanism enables the Web to provide a generic migration service to such component systems. In this way, the Web can become a distributed platform, enabling distributed systems to build their own specific services on top of the Web’s generic services. For example, system-specific messages between components can be routed to individual resources (components) irrespective of where the resources are located, using the generic services provided by the migration mechanism.

Optimising the network through network traffic profiling

Deciding which content to migrate and when can optimise both the performance of a server, and a network as a whole. Currently, certain network technologies and service level agreements (SLAs) with network providers insist on the network user specifying the expected quality of service of the network at certain times of the day. For example, Frame Relay can ensure a certain throughput to the user over a short period of time by guaranteeing a committed information rate (CIR). This CIR is the rate which is, on average, available to the user.

Determining the CIR is a difficult process and involves a good knowledge of the local traffic. The network manager has to plan for the expected traffic, keeping in mind that at very busy times he does not have the same throughput and availability of capacity above the CIR for “bursty” traffic. Traffic profiling is very important in such networks, whereby the traffic is monitored in order to determine the quality of service required. Research by the members of the author team is developing a methodology for profiling traffic in this way. It has been determined that while overall network traffic may be variable over the short term, over time it only increases. The SLAs, therefore, can provide the business case for introducing content migration as a means of balancing the network and staying within the CIR. With a transparent migration mechanism built into a company’s intranet, software resources can be migrated to balance the load not just across servers, but across the network. A traffic profiling system can be used to monitor the traffic on a company’s network. If network traffic has increased at a particular

point, resources can be migrated to ease the flow of traffic at the bottleneck. If the traffic is too great only at certain times of day, the profile will show this, and the resources can be migrated back and forth according to the time of day.

Conclusion

This paper has examined the various issues involved in developing a practical migration mechanism for the Web. It has identified the requirements of such a mechanism, and examined some of the different approaches that can be used to implement a migration mechanism with respect to these. However, no current migration system meets these requirements, largely because they are not content-neutral. Therefore, the authors are currently working on a migration mechanism that will meet these requirements, and thus could form part of the Web's infrastructure. A transparent, search-based, resource-level migration mechanism for the Web, combined with existing distributed component architectures and sophisticated network traffic profiling techniques should optimise both a server and the network, and can provide a new class of services to users. While the Web is currently a distribution system, the integration of a migration mechanism can provide the Web with the services it needs to offer to become a ubiquitous distributed system.

References

- Berners-Lee, T. and Connolly, D. (1995), "HyperText Markup Language – 2.0", RFC 1866, MIT/W3C, November.
- Berners-Lee, T., Fielding, R. and Frystyk, H. (1996), "Hypertext Transfer Protocol – HTTP/1.0", RFC 1945, MIT/UCS/UC Irvine, May.
- Berners-Lee, T., Masinter, L. and McCahill, M. (1994), "Uniform resource locators (URL)", RFC-1738, CERN/Xerox/University of Minnesota, December.
- Briscoe, R.J. (1997), "Distributed objects on the Web", *BT Technology Journal*, Vol.15 No.2, April, pp.158.
- Ingham, D., Caughey, S. and Little, M. (1996), "Fixing the 'broken link problem': the W3Objects approach", in *Fifth International World Wide Web Conference*, 6-10 May, Paris.
- Franklin, S. and Graesser, A. (1996), "Is it an agent, or just a program?: A taxonomy for autonomous agents", in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer Verlag, New York, NY, Berlin, 1996, <http://www.msci.memphis.edu/%7Efranklin/Agent-Prog.html#agent>
- Kappe, F. (1995), "A scalable architecture for maintaining referential integrity in distributed information systems", *JUCS*, Vol. 1 No. 2, February, pp. 84-104.
- Mic Bowman, C., Danzig, P.B., Hardy, D.R., Manber, U. and Schwartz, M.F. (1995), "The harvest information discovery and access system", *Computer Networks and ISDN Systems*, Vol. 28, pp. 119-25.
- OMG (1995), "The Common Object Request Broker: Architecture and Specification, Revision 2.0", Object Management Group.
- Orfali, R., Harkey, D. and Edwards, J. (1996), *The Essential Client/Server Survival Guide*, John Wiley & Sons, New York, NY, and Chichester.
- RM-ODP (1995), "Open distributed processing reference model (RM-ODP)", ISO/IEC DIS 10746-1 to 10746-4, 1995. <http://www.iso.ch:8000/RM-ODP/>
- Sollins, K. and Masinter, L. (1994), "Functional requirements for uniform resource names", RFC 1737.
- Tanenbaum, A.S. (1996), *Computer Networks*, 3rd ed., Prentice Hall, Englewood Cliffs, NJ.

Internet-based security incidents and the potential for false alarms

*M.P. Evans and
S.M. Furnell*

The authors

M.P. Evans is a Research Student and S.M. Furnell is a Research Co-ordinator, both at the Network Research Group, School of Electronic, Communication and Electrical Engineering, University of Plymouth, Plymouth, UK.

Keywords

Computer security, Internet, WWW, Hacking

Abstract

The paper explains the background to experimental work that was conducted with the aim of measuring aspects of the WWW (specifically the average lifetime of a web link and the impact of the "Millennium Bug"), but which inadvertently caused two perceived security breaches on remote systems. The paper explains the nature of these incidents and considers why, when over 700,000 IP addresses were randomly sampled in the experimental study, only two sites considered the activity to be an attempt to breach their security. It is concluded that, while the appropriate protection of Internet-based systems is undoubtedly of importance, the problems experienced during the experimental study suggest a lack of uniformity in what different organisations will class as a security breach.

Electronic access

The current issue and full text archive of this journal is available at
<http://www.emerald-library.com>

Introduction

The Internet and the World Wide Web (WWW) represent two of the most significant technological developments of the last century. The Internet offers a previously unimaginable potential for connectivity – the possession of two IP addresses enables a seemingly direct connection between two systems, no matter where they may be physically located. With network connectivity and appropriate software utilities, it is possible to determine the existence of a remote system (and, to some extent, the services it can provide), even if you cannot actually log into it. From a security perspective, this can represent a problem, as the mere knowledge of a system's existence serves to make it a potential target.

The paper describes the problems posed by Internet-based attacks and the resulting attitude that they demand on the part of network security administrators. The discussion then proceeds to consider the problem that this effective state of "connection paranoia" may represent for normal Internet users wishing to conduct innocent network activities. This is supported by the example of an experimental study into Web site longevity that the authors attempted, but which was complicated by inadvertent security side effects.

Internet security issues and examples

Computer-based crime and abuse has been recognised as an unfortunate side-effect of the information technology revolution for many years, with computer hackers (crackers) and viruses representing the most widely recognised causes of the problem (Audit Commission, 1998; Furnell and Warren, 1996). However, the Internet has significantly enhanced the threat posed by attackers, giving rise to a new series of opportunities for abuse. For example, a useful tool in the hacker arsenal is the port scanner, which enables the inspection of a remote system to determine what software and services it is running. This knowledge can be useful in facilitating a more direct attack if known vulnerabilities of the discovered software can then be exploited. Probably the most well-known example of a scanner program is the

Security Administrator Tool for Analyzing Networks (SATAN), the rationale for which is described by Farmer and Venema (1993). SATAN, in common with other scanner software, offers the facility to find a machine or network, find out what services are being run and then automatically test those services for known security holes. As the full name suggests, SATAN offers a useful tool to system administrators wishing to ensure the security of their network. However, the public availability serves to make it an accessible tool for hackers as well. SATAN has been followed by a wealth of other tools, such as NetInfo, PortPro, IPprober and HackTek, from both the commercial and hacker communities.

Even if an attacker cannot directly enter a system, they can often still cause problems through denial of service (DoS) attacks. Such an attack is one in which a target system is rendered inaccessible or unusable and will generally involve the consumption of a system's resources (such as memory, storage space and/or network ports), such that it is unable to provide adequate service for its legitimate users. At a minimum, the end result can be inconvenience for legitimate users attempting to use the system or wishing to gain access to it. Such incidents represent a growing problem on the Internet and account for a significant proportion of reported security problems (CERT, 2000). The attacks themselves can take many forms and are frequently system-specific, taking advantage of known vulnerabilities on a particular target platform (e.g. running a particular operating system, Web server or other system software). However, some generic categories of DoS attack can also be identified and two Internet-based examples are briefly described below (Escamilla, 1998).

(1) *Ping of death*. Relies on a flaw in some TCP/IP stack implementations. The attack relates to the handling of unusually and illegally large ping packets (which some systems, e.g. Windows NT and 95, can generate). Remote systems receiving such packets can crash as the memory allocated for storing packets overflows. The attack does not affect all systems in the same way, some systems will crash, others will remain unaffected.

(2) *SYN flooding*. Exploits the fact that establishing a connection with the TCP protocol involves a three-phase handshake between the systems, as follows:

- connecting host sends a SYN packet to the receiving host;
- receiving host sends a SYN|ACK packet back;
- connecting host responds with an ACK packet.

In a SYN flood attack, an attacking host sends many SYN packets and does not respond with an ACK to the SYN|ACKs. As the receiving host is waiting for more and more ACKs, the buffer queue will fill up. Ultimately, the receiving machine can no longer accept legitimate connections.

The facility to launch such attacks can be found as a "feature" of several cracker toolkits, such as HackTek. Given that they can be automated in this way, the mounting of such attacks does not require any skill or expertise on the part of the hacker (indeed, more dedicated hackers refer to those who rely upon such techniques as "script kiddies"). As such, it has been conjectured that around 90 percent of hacking is conducted by people using such methods (Akass, 2000). This is not to underestimate the serious consequences that attacks such as denial of service may have. For example, in the context of a system used for a sensitive application, such as providing access to patient records or controlling direct care provision in a healthcare environment, any unavailability or performance degradation could have significant consequences.

The sheer range of companies and organisations now represented on the Internet and the WWW means that there are a significant number of high-profile targets for potential attackers. One such organisation is the Pentagon in the United States, which experienced a total of 5,844 recorded attacks in 1998. Although this is itself significant (averaging over 16 attacks per day), the number recorded in 1999 was significantly higher and, by November, well over 18,000 attacks had been identified (Daily Telegraph, 1999). More recently, a distributed DoS attack was experienced by a series of major Internet sites in February 2000. The attacks affected a number

of notable and popular sites, including Amazon.com (books), eBay (online auctions) and CNN (news) and had a significant impact. For example, it was reported that, within a few minutes, the Amazon.com Web site became 98.5 percent unavailable to legitimate users (McCullagh and Arent, 2000). In addition, Keynote, a US-based Internet monitoring company, reported that the average performance of the Internet was degraded by “as much as 26.8 percent” (Keynote, 2000). These statistics serve to reinforce the significance of the problem that DoS attacks can represent.

As a consequence of factors such as those described above, many organisations are sensitive to the threat of Internet-based attacks and take measures to guard against them. Success here relies upon being able to accurately detect the signs of an attempted attack in progress. However, as the next section illustrates, it is not always possible to reliably differentiate between attacks and other forms of network activity.

Measuring the Web – a problematic study

The authors have first-hand experience in causing Internet security alerts – although in an entirely innocent context, through the conduct of Web-based experimental research unrelated to security. The nature of the intended experiment, and the problems that subsequently arose, are described in the sections that follow.

Experimental background and procedure

The experiment that caused the problems was involved in measuring aspects of the Web. Specifically, the experiment was designed to determine both the average lifetime of a Web link (that is, the average period of time before the resource pointed to was removed), and the impact of the “Millennium Bug” on the Internet. Both of these activities were conducted as part of a wider research programme relating to Web-based content migration (Evans *et al.*, 1999).

The principal aim of the experiment was to determine the average lifetime of a Web link. To do this, it was necessary to collect a large

sample of links at random. Each link would then be tested periodically, and the date and time would be recorded if and when the link failed (i.e. the resource pointed to by the link could no longer be found). To ensure the randomness of the links, the authors attempted to compile a database of Web servers from a large list of Internet servers chosen at random. Once the list had been compiled, the intention was to let a Web crawler search through the various HTML documents on the Web servers, and choose for itself a set of links at random. This would ensure no bias had crept into the link selection process. Unfortunately, however, the experiment never reached this stage.

A secondary aim of the experiment was to determine the impact of the “Millennium Bug” on the Internet. The list of randomly selected servers was being compiled between October and November 1999. As such, it would be trivial to extend the experiment and periodically test the state of these servers after midnight on 1 January 2000. Although not related to the primary aim of the experiment, this would be interesting research for little cost.

The intended experiment comprised four different stages:

- (1) compile a list of random servers;
- (2) from this list, compile a list of random web links;
- (3) periodically determine the state of each server;
- (4) periodically determine the state of each link.

However, only the first stage was ever reached, as several unanticipated side effects caused two unintentional security incidents, which forced the experiment to end prematurely. The side effects were a direct result of the design of stage 1 of the experiment interacting unpredictably with a server’s firewall.

In stage 1, a list of random Web servers had to be compiled. This was achieved by randomly generating an IP address and sending a simple HTTP HEAD message on port 80 to attempt to retrieve the header of the default HTML page of the server. If a response was received, the server’s IP address and domain name were recorded as belonging to a Web server. If no response was received, however, the machine was pinged. If a response was received, the

machine's IP address and domain name were recorded as belonging to an Internet server (no attempt was made to determine which type of Internet service the machine was providing). This allowed the experiment to determine the effect of the Millennium Bug on the Internet in general and on the Web in particular. If no response was received from the ping, the IP address was noted as being dead, and played no further part in the experiment.

Because of the unpredictable nature of the network at the University, the experiment was designed to compensate for any network problems that occurred. For example, the network would sometimes go down for a few seconds or several minutes before returning to normal. At other times, its speed dropped significantly, due to the network loading of the University's LAN. Because of this, some ping messages seemed to take a long time before a reply was received; equally, some HTTP requests had to be sent more than once before a reply was received. To ensure that a server really was down when no reply was received, the HTTP HEAD request was sent more than once, and both the Time To Live of each ping, and the number of pings sent, was increased to allow for delayed responses.

The experiment was designed as a multi-threaded application, with 100 threads operating at a time, each of which contacted one server at random. The design was fully tested on the University's servers before being allowed to sample the whole Internet. However, over time, it became apparent that the experimental procedure was resulting in perceived security problems and, during the course of two months, two formal complaints were received from organisations whose systems had been randomly targeted. The details of these incidents, and the remedial actions that were taken, are described in the sections that follow.

Security incident one

Table I contains the text of an e-mail received from the administrators of one of the randomly selected systems. It should be noted that elements of the figure have been edited to preserve the anonymity of the specific machines involved at the authors' site and of the affected remote domain. As such, organisational name

Table I E-mail received concerning first perceived intrusion

We have detected unfriendly network activity, directed at our machines, from 141.163.xx.xx [xx.xx.plymouth.ac.uk]. The activity, which began at 19:04 EDT (GMT-0400) on October 27, 1999 was a port scan (137) and pinging of many addresses in our subnets (xx.xx.xx.xx, xx.xx.xx.xx).

This type of activity is not desired on the [Deleted] domain and is monitored frequently. Please advise your system managers and users that this activity should stop immediately.

[Deleted]
Computer & Network Security

references have been deleted and address elements have been replaced with "xx" where appropriate.

From the perspective of the remote system, it is clear that the activity of the program was considered comparable to that of port scanning tools such as SATAN. However, in the context of the experimental study, the use of port 137 (the NetBIOS name resolution service) was not a feature that had been explicitly included in the program. Rather, it is a feature of the Windows NT Server (the operating system that was used to host the experiment) that NetBIOS is used to resolve a name, followed by the Internet DNS, in response to a call to the "gethostbyaddr()" function. This is the default behaviour of the operating system.

Although completely innocent in this case, port scans are widely used by hackers as reconnaissance, in an attempt to determine the services the victim's servers are providing. However, in this case the firewall software would have received one NetBIOS call to port 137, and one subsequent HTTP request to port 80 (although the latter was not mentioned in their e-mail communication). Further, the "pinging of many addresses" (ping sweep) that was detected listed just two IP addresses that were pinged. Both addresses were selected completely at random, and were not part of the same sub-net, or even looked as if they were remotely related. It was an unfortunate coincidence that both were pinged at the same time, and both happened to be owned by the same organisation.

Security incident two

The experiment was remounted, with NetBIOS support disabled on the originating NT system,

and the ping configuration altered such that each ping packet had a timeout value of one second, and at most only six packets would ever be sent to one machine. The experiment was restarted, and for a time all went well. In fact, 700,000 different IP addresses had been tested over a period of two weeks, when the second and final inadvertent security incident occurred. Table II presents the e-mail message received in this case. Note that the IP addresses listed have again been altered in order to hide the identities of the machines involved.

At this point, it was decided that the experiment had attracted too much adverse attention and the University's Computing Service was understandably concerned that further problems could arise if it was to continue. As such, the mutual decision was taken to discontinue this element of the study.

The unfortunate situation in this case was the fact that the incident reported differed greatly from the behaviour of the experiment under testing. A total of 338 ping messages were allegedly received in five seconds, when the code was explicitly written to send only six. Indeed, the network traffic generated by the experiment had been extensively monitored before the experiment was restarted to ensure

that no more than six packets were sent to any one machine. The University's network was used for the test, and indeed, only six packets were ever sent to each machine. However, the incident reported 338 such messages. Whether this was a fault in the experiment or in the firewall on the remote machine that reported the incident is impossible to determine, as the experiment had to be discontinued and it is doubtful that the "victim" would be willing to share their firewall's configuration details.

Discussion

In both incidents, the receipt of the complaint was immediately followed by corrective action and a written explanation of the experimental context. This was considered satisfactory and defused the possibility of further action.

Reflecting upon the experiences, it could be argued that in the first incident, the remote firewall software used was a little overzealous. The "incident" comprised one name resolution request to port 137, one HTTP request to port 80 and several ping messages on two separate servers with widely differing IP addresses. It is debatable whether this should be considered to represent a threat. It should also be remembered that a contributing factor in the first incident was that the experimental software performed an unexpected task (i.e. a NetBIOS call) as a consequence of asking it to perform an intended function (i.e. a name resolution). In a sense, it could therefore be argued that the program author was the victim of an inadvertent "Trojan Horse" effect. Without this prior action occurring, it is possible that the respondent organisation may not have perceived the subsequent pings to be part of a hostile attack.

In the second incident, the duration of each ping was only five seconds, which intuitively would not suggest that a Denial of Service attack was intended. It could be argued that this might have been an initial assault, designed to overload the firewall system and thereby enable exploitation of some other vulnerability. However, the fact that no further activity would have been apparent from the source IP address should have provided an indication that this was not the case.

Table II E-mail received concerning second perceived intrusion

Intrusion attempt report

We have noticed the following behaviour originating from IP addresses under your control:

ICMP denial of service attempt

The activity took place at approximately:

Dec 7 03:07 GMT

We consider this/these unauthorized attempt(s) to access our networks as malicious in nature and hereby request that you take steps to identify the person(s) involved and arrange for this activity to halt immediately.

Here are some samples of the activity in question:

03:07:41.035397 141.163.xx.xx	xx.xx.xx.xx: icmp: echo request
03:07:41.036032 141.163.xx.xx	xx.xx.xx.xx: icmp: echo request
03:07:41.038980 141.163.xx.xx	xx.xx.xx.xx: icmp: echo request
03:07:41.039568 141.163.xx.xx	xx.xx.xx.xx: icmp: echo request
03:07:41.042556 141.163.xx.xx	xx.xx.xx.xx: icmp: echo request
03:07:41.043138 141.163.xx.xx	xx.xx.xx.xx: icmp: echo request

338 instances in five seconds

We further request that you reply back to us with the resolution achieved in this matter.

In both incidents, the ping feature had been disabled by the firewall (a relatively standard practice, which is intended to guard against attacks such as "Ping of Death"). This meant that the experiment continued sending ping requests in an attempt to determine whether the IP address was a live server or not, while the firewall silently monitored the requests, yet did not respond. In the absence of a response, the experimental software was unintentionally entrapped into appearing as a security threat.

With hindsight, however, it can also be argued that, from a security perspective, the practical approach taken by the experimental study was ill conceived. To select an IP address at random and then attempt to determine the state of the server can be seen to have the potential for mis-interpretation by a security conscious organisation. From the organisation's perspective, such a stream of traffic would have no obviously legitimate purpose and, therefore, by default would be regarded as suspicious. However, the fact that only two organisations flagged a problem during the period of the experimental study (during which over 700,000 random addresses were targeted in this way), gives a very strong indication that organisations are not monitoring their security to a consistent degree. If it is argued that the two complainant organisations were correct to interpret the network activity as attacks, then it could also be considered that the other organisations were failing in their network security strategy. This assertion must, of course, be offset against the fact that different organisations will be dealing with systems and data of different levels of sensitivity, and therefore, in some cases, the required level of security may legitimately be lower. Having said that, it is unlikely that in a random sample only two organisations had data that they would consider sensitive.

The question remains, however, as to how to effectively monitor the growth of the Internet without upsetting somebody's security policy. As e-commerce continues to grow on the Web, security is becoming more and more of an issue, with the result that firewalls are being configured ever more tightly. As has been shown, this can have the effect of seriously derailing entirely innocent applications, and has the potential to cause serious harm to the reputation of those involved (in this case, the

University was advised that unless the experiment was discontinued, its network connection would risk being terminated).

In view of the practical experiences, it is worth re-examining the Pentagon attack figures cited earlier in the paper. Closer investigation reveals that the Pentagon's definition of an attack includes activities such as port scans and pings (Wayner, 1999). As such, the level of genuine abuse may not be as significant as first suggested by the bare statistics alone, as many of the incidents recorded may have been the result of activities that were not intended to breach security. This can be regarded as a counter-argument to the commonly held belief that the majority of computer-based crime goes unreported, due to fears of adverse publicity on the part of the affected organisations (Nycum and Parker, 1990). In this case, organisations may be over-stating their vulnerability to abuse. In addition, there has been at least one legal ruling (by the Norwegian supreme court) stating that probing of systems on the Internet, using techniques such as port scans, should not be considered illegal (Jones, 1998).

It can be argued that all parties involved in the practical incidents described emerged as losers from their experiences. The authors were unable to proceed with a potentially interesting experiment, whilst the remote organisations had effectively wasted resources in responding to false alarms. As such, there appears to be the need for some form of protocol through which applications such as the experiment detailed here can safely query a server on the Internet without upsetting its security arrangement. Such a protocol exists for the Web in the form of the Robot Exclusion Standard (Koster, 1994). This is a mechanism through which a Web server can define the permissible behaviour of "visiting" software agents, such as a search engine's Web crawler. These agents attempt to index the contents of a Web server, but have the potential to cause unwelcome side effects. For example, they may flood the server with too many requests, or attempt to index areas in which they are not welcome, either for privacy reasons, or because their presence fools the server into thinking that they are a genuine user. To prevent this, the standard lets the server owner specify clear boundaries of good behaviour which the Web crawler is expected to

adhere to. The boundaries are encoded in a text file called *Robots.txt*, which a Web crawler should parse upon arrival at the server. Although the standard cannot enforce the behaviour of a Web crawler, it provides an implicit contract between server owner and Web crawler designer. Breaking the contract can lead to the exclusion of all traffic from the sub-network from which the Web crawler originated, or even legal disputes (Pallmann, 1999). As such, the Robot Exclusion Standard could make an ideal model from which to develop similar protocols for measuring the Internet. With so many different security policies in existence, it makes sense for a server to publish its policy of acceptable behaviour, rather than expect any visiting software agents to guess what that policy might be.

Conclusions

The provision of appropriate protection for Internet-based systems is undoubtedly of importance. It is necessary for systems to have “frontline” defences in order to detect potential abuse and reduce the possibility of successful system penetration. From this perspective, the organisations that identified and responded to the activity of the experimental software can be commended for having effective security monitoring procedures that should also enable detection of genuine attacks. The fact that, in the cases described, the monitoring software caused false alarms can be excused in a security context, as this is preferable to allowing an intruder to penetrate or disrupt the system.

Incidents such as Denial of Service attacks represent a significant threat to Internet-based systems and, while they do not represent a direct threat to the confidentiality or integrity of data, they may be employed as a precursor to a more direct form of attack. In addition, the lack of system availability may itself represent a significant threat to individual or organisational well-being in many scenarios. As such, it is legitimate and advisable for organisations to take appropriate steps to protect their assets from such attack. However, there is also a clear need for experimental research to be performed on the Internet. Measurements on the size of the Internet and its growth are not just of

academic relevance. Web masters and network managers need to know how much network traffic to expect, while business leaders need to know the importance of the Internet to their business. The techniques used to perform such measurements, however, share certain characteristics with those used by hackers. There is, therefore, the potential to trigger security alerts for innocent reasons, as the authors have discovered. This finding calls into question whether the number of “security incidents” logged by certain organisations actually represent a realistic indication of their vulnerability to attack. The experiment discussed previously was applied to over 700,000 Internet addresses, yet only two security incidents were flagged and followed up, demonstrating a lack of uniformity in the security policies of different organisations and what they class as an attack.

In view of the above, the security needs of the Internet must be balanced with the experimental needs of the research community (and the activities of legitimate bots performing necessary services), if only to prevent future misunderstandings that could potentially lead to more embarrassing outcomes than those discussed here. The definition of a standard enabling server owners to define their acceptable behaviour policy could prevent such situations from occurring, and lead to a more realistic measure of security incidents, reducing public fear caused by potentially exaggerated attack statistics.

References

- Akass, C. (2000), “On the straight and narrow – not”, *Personal Computer World*, February, p. 57.
- Audit Commission (1998), *Ghost in the Machine*, Audit Commission Publications, February, ISBN 1-86240-056-3.
- CERT (2000), “CERT “ Advisory CA-2000-01 Denial-of-Service Developments”, CERT Coordination Center and the Federal Computer Incident Response Capability (FedCIRC), 3 January, <http://www.cert.org/advisories/CA-2000-01.html>
- Daily Telegraph (1999), “Pentagon under cyber-seige”, *The Daily Telegraph Connected Supplement*, 11 November, p. 2.
- Escamilla, T. (1998), *Intrusion Detection*, Wiley Computer Publishing, ISBN 0-471-29000-9, New York, NY.

- Evans, M.P., Phippen, A.D., Mueller, G., Furnell, S.M., Sanders, P.W. and Reynolds, P.L. (1999), "Strategies for content migration on the World Wide Web", *Internet Research*, Vol. 9 No. 1, pp. 25-34.
- Farmer, D. and Venema, W. (1993), "Improving the security of your site by breaking into it", <http://www.fish.com/~zen/satan/satan-demo/admin-guide-to-cracking.html>
- Furnell, S.M. and Warren, M.J. (1996), "Computer abuse: vandalising the information society", *Internet Research*, Vol. 7 No. 1, pp. 61-6.
- Jones, C. (1998), "Let the Web server beware", *Wired News*, 23 December, <http://www.wired.com/news/politics/0,1283,17024,00.html>
- Keynote (2000), "Denial of service attacks this week degraded Internet performance overall according to Keynote", Keynote Press Release, 12 February, <http://www.keynote.com>.
- Koster, M. (1994), "A standard for robot exclusion", *The Web Robots Pages*, <http://info.webcrawler.com/mak/projects/robots/norobots.html>
- McCullagh, D. and Arent, L. (2000), "A frenzy of hacking attacks", *Wired News*, 9 February, <http://www.wired.com/news/print/0,1294,34234,00.html>
- Nycum, S.H. and Parker, D.B. (1990), "Prosecutorial experience with state computer crime laws in the United States", in Grissonnanche, A. (Ed.), *Security and Protection in Information Systems*, Elsevier Science Publishers B.V., North-Holland, pp. 307-19.
- Pallmann, D. (1999), *Programming Bots, Spiders, and Intelligent Agents in Microsoft Visual C++*, Microsoft Press, ISBN 0-7356-0565-3.
- Wayner, P. (1999), "Hacker 'attacks' on military networks may be closer to espionage", *New York Times*, 8 March, <http://www.nytimes.com/library/tech/99/03/cyber/articles/08defense.html>

The Resource Locator Service: Fixing a Flaw in the Web

M.P.Evans and S.M.Furnell

Network Research Group
Department of Communication and Electronic Engineering, University of Plymouth,
Plymouth, UK.

Abstract

The architecture of the World Wide Web has scaled beyond its original expectations, but problems are now emerging that could undermine its effectiveness as an information system, and restrict its future growth. Nearly 30% of all web pages experience link rot, DNS domain names are rapidly running out, and older web pages are deleted without being archived, leading to the loss of potentially important information. These problems are caused by the URL and its reliance on the Internet's DNS for its namespace, which we argue represent a serious flaw in the web's architecture. In this paper, we present a new web-specific name resolution service that has been designed to address these problems. Called the Resource Locator Service, it offers an unconstrained namespace, and a mechanism for transparent resource migration that can dynamically locate static resources across time and space.

Keywords: Referential integrity; Resource migration; Link rot; Temporal references; Web namespace

1. Introduction

The World Wide Web (web) was designed by Tim Berners-Lee as a social creation rather than a technical one [14]. The ease with which its users can publish information as well as read it, combined with its exponential growth, has made it a social platform from which ideas and concepts emerge at an ever-increasing rate. However, the sheer volume of users and information has applied enormous pressure on its architectural foundations, which was not foreseen during its development. As its size continues to grow exponentially, increasing pressure is placed on its architecture, such that any flaw will become a major weakness in the system. With the web's role in society becoming increasingly important, and with the development of new access devices such as Personal Digital Assistants increasing the number of users, it seems appropriate to address any flaws before they disrupt the system.

In its present design, the most serious flaw in the web's architecture currently stems from the design of the Uniform Resource Locator (URL), which is used to reference a web resource. The URL has proven to be an unfortunate means of referencing a resource on the web, and its technical limitations are well documented [8, 14, 18, 27, 30, 35-36, 39]. It sits uneasily between the machine world of the web's architecture and the human world of the User Interface: the machines need the URL to be syntactically consistent and constrained to tell them *where* a resource is, whereas humans need it to be intelligible and memorable to tell them *what* the resource is. For the purposes of this paper, we have focused on three key problems that are inherent within the URL's design, which together could threaten the web's development if left unchecked:

1. *Link Rot* – the URL incorporates a server’s hostname in order to provide a name for a resource. When the resource migrates to a different host, it must use a new URL that incorporates the new hostname. This causes all hyperlinks that use the old URL to break, or ‘rot’. Currently, 28.5% of web pages suffer from link rot, with 5.7% of all links broken [38] and an average of 5.3% of links in search engines also broken [21]. The informational content and overall usefulness of the web will decrease as links become less reliable.
2. *Shrinking Namespace* – a URL not only defines a resource’s location, it is also used as its name. Any company that wants to be remembered needs a memorable name, and the trend on the web has been to name a company after a memorable hostname to create a memorable URL. However, the URL is based on the aging Domain Name System (DNS) of the Internet, and the namespace this provides is running out. In March 2000 alone, new hostnames were being requested at a rate of nearly one a second, and some 14,322,950 distinct hostnames have been registered just for web server use [29]. The problem is exacerbated by copyright and trademark issues regarding the ownership of certain URLs, and the centralised nature of the DNS, making the URL in its role as a resource’s identifier, the “web’s achilles heel” [3].
3. *Lost History* – the web is designed for society, but crucially it neglects one key area: its history. Information on the web is *today’s* information. Yesterday’s information is deleted or overwritten, with no consistent means of searching through archived material other than manual navigation through a web site in the hope of finding archived material. The URL is a spatial identifier only, unconcerned with the temporal ordering of the web’s resources, and so prevents the consistent retrieval of archived information [22].

In this paper, we examine existing solutions to these problems and highlight their weaknesses when confronted with a system the size of the web. We argue that the architecture of the web itself is flawed, and that solutions built on top of a flawed architecture cannot work. As such, we present a new approach with the design of the Resource Locator Service (RLS), which effectively addresses these problems by replacing the DNS with a name service designed specifically for the web.

The remainder of the paper is presented as follows: Section 2 discusses the background to the problem, and related work that has tried to provide a solution. Section 3 provides an overview of the RLS, while section 4 presents the design in much greater depth. Finally, sections 5 and 6 discuss the RLS in operation, while sections 7 and 8 present the results of performance measurements that have been taken from a prototype, which has been developed to demonstrate the effectiveness of the design, and discuss issues and further work that remains to be done.

2. Background and Related Work

2.1. Solutions to Link Rot

The use of the URL as a means of identifying a resource has caused all links to be inherently brittle, as once a resource moves to a new location, the link breaks. Designing a system for the web whose identifier does not change when the resource migrates (i.e. a location-independent identifier) will help to prevent link-rot. Such systems exist on the web and can be classified as using one of five approaches:

1. *The Chain Approach*

A forward reference is left behind on the machine that the resource has migrated from, pointing to the new location. Although arguably optimal in terms of network traffic overhead [14], this approach can lead to forward references outnumbering

resources. Various shortcut operations can limit the length of the chain of forward references, but this approach is still inherently brittle, as locating your resource is dependent upon the state of someone else's server. Also a resource can only migrate onto a server that supports this approach. Examples include W3Objects [14].

2. *The Callback Approach*

A database of all the links on the web is maintained. Each time a resource migrates, the database is updated and calls back all documents that contain a link to the resource, enabling each document to update its links. This approach guarantees referential integrity, as it is modelled on database technology. However, the web is not a database, and any server on the web may be down at any time. As the database must store all updates to servers that are down, it would eventually be overwhelmed by the number of pending updates [6]. This approach also requires the documents to be intelligent enough to remove links identified as broken, and so is not backwards compatible with the web's existing architecture. Examples include the HyperG system [18] and Atlas [30].

3. *The Search Approach*

Whenever a resource needs to be located, a network-wide search is performed, with a flooding algorithm used to guarantee that all servers are queried. Although reliable, such an approach produces too much network traffic overhead for use on the web, and is the least optimal of all the approaches [14]. No examples currently exist for web-wide resource location through search.

4. *The Name Server Approach*

A set of distributed name servers are used that manage a resource-identifier/location mapping. The name server is queried using the identifier of the resource in much the same way that a machine's IP address is determined through its hostname using the DNS [25, 26]. However, a name server system is essentially a distributed database, whereas the web is a federated system, and so locating the correct name server without breaking the web's existing architecture presents a significant challenge. Examples include the Handle System [39], and the Resolver Discovery Service (RDS) [36], both of which break the web's existing architecture.

5. *The Lecturing the User Approach*

Not a technical approach, more a philosophical one. Berners-Lee and others have argued that a URL need not break if considered thought is given to its design [2]. However, despite the numerous technical arguments against this viewpoint, it is people who create URLs and people who are notoriously bad at consistent regular maintenance. Ultimately, as broken links on the W3C web site itself testify (e.g. the link <http://discuss.w3.org/mhonarc/w3c-tech/threads.html> on the document located at <http://www.w3.org/MobileCode/Workshop9507/> is broken), lecturing the user will be ineffective at best.

2.1.1 *Semantic Ambiguity*

Although each of the five approaches provides its own solution to the problem of link rot, the semantics of the link and what it references are left in an ambiguous state. Referential integrity can ensure that links always reference the same resource, but what happens if the content contained within the resource changes? Should the semantics of the link require the content to persist for the lifetime of the resource, thus requiring a new resource and identifier to be created each time the content changes; or should the semantics be defined such that new content simply overwrites existing content? The former option will preserve all content, but will lead to an explosion of new resources, each with its own distinct identifier. Web sites that contain frequently changing content, such as daily news sites, will generate many new

resources, making linking to the site virtually impossible. Conversely, the latter option controls the number of resources but destroys information. Links will only be able to reference the web *site*, rather than specific information on the site, requiring the user to search manually for the story within the site's archives (if they exist). Although this problem of semantic ambiguity exists in the web's current architecture, the design of a new name service is a suitable opportunity for the ambiguity to be resolved.

2.2. Solutions to the Shrinking Namespace

Since the birth of the web, the number of domain names registered has exploded exponentially, leading to the number of memorable names shrinking rapidly. Companies that register domain names without actually using them in order to resell them for a profit exacerbate the problem by driving up the price of the remaining names. Furthermore, name disputes are becoming increasingly common, as the rights to the remaining names are fought over by companies and organizations with similar trading names. The Internet Corporation for Assigned Names and Numbers (ICANN) is the organization responsible for assigning Internet names, and acts as the central registrar for domain names on the web. ICANN has a defined policy for resolving domain name conflicts, called the 'Uniform Domain-Name Dispute Resolution Policy' [16], which attempts to resolve the issue of two parties fighting over the same name. However, before the policy can be invoked, one party must first file a complaint in a court of law. This is not an elegant solution, and with the number of available domain names dwindling, the problem can only get worse.

To resolve the problem, ICANN is currently examining ways to extend the top level domain name space. A Top Level Domain name (TLD) is the last part of a domain name (e.g. .com, .org, etc.). ICANN has recently extended the original list of seven (excluding country codes) to include new names such as .biz, .coop, and .aero, etc [13]. However, this must be seen as a short-term solution, as it simply constricts the same problem to vertical commercial and organizational domains. Equally, there is no guarantee that the new top level domains will be used, as companies are currently fighting to use the .com TLD over all other alternatives. Currently, 82.8% of all registered domain names use .com [20], largely because it is perceived as being associated with the web in people's minds far more strongly than any other TLD [37]. As such, the competition for .com names will still remain, regardless of how many new TLDs are introduced.

An alternative proprietary solution can be found in the RealNames system [31], which provides an alternative namespace to that of the DNS, and is used by various web portals including AltaVista, MSN, Google and LookSmart. RealNames uses *Internet Keywords* as 'human friendly identifiers' [23] that are registered by a company or organization usually associated with that name (e.g. 'Ford'). The RealNames system maps the human friendly identifier onto a company's web site, enabling a user to navigate to the site using the brand name of the company. In addition, it allows more than one party to register a web site under the same Internet Keyword, presenting several links to the user (either through an affiliated web site, such as AltaVista, or an affiliated browser, such as Microsoft's Internet Explorer 5) when a shared name is entered. In effect, it can be seen as occupying the middle ground between a naming scheme and a search engine, indexing Internet Keywords rather than every word in a web document, and so makes search results more reliable than a general web search. However, it is not a true architectural solution, as the same identifier does not uniquely identify a specific resource, leaving it to the user to manually select the most appropriate resource from a list. As such, an Internet Keyword cannot be used by the web as a machine-readable identifier, and so cannot replace the URL. Furthermore, it does not open up the namespace, as it is a proprietary solution that enables the RealNames company to determine what is and what is not a suitable Internet Keyword. The company is also in a position to limit the number of times that an Internet Keyword can be used by users to just 1000 times a year, at a cost of \$100 per Internet Keyword per year [32].

The official solution is to use Uniform Resource Names (URNs) rather than URLs. URNs are designed to be permanent identifiers that identify a resource through a location-

independent name, thus simultaneously removing the dependence on the DNS and providing a solution to link rot. However, URNs have been on the agenda since 1992, and despite many short-term solutions [7, 27], no architectural method of providing URN to URL resolution has been developed. Worse, URNs are designed as machine-readable identifiers only [36], and so ignore the shrinking namespace problem completely as they are not designed for human use.

2.3. Solutions for Archiving the Web

Because the URL is a spatial locator and has no means of referencing a resource according to its time of creation, the web is always stuck in the present. A user can manually locate an archived version of a resource using the textual cues contained within a web page, but a web crawler cannot, as it does not understand the text. Existing solutions to this problem are again proprietary and unfocused in their approach. For example, the Internet Archive project [17] was begun in April 1996 by Brewster Kahle to literally archive the entire Internet. However, access to the archive is free only to researchers, students and not-for-profit organizations, and only if a research proposal indicating the need for access is first submitted and approved. Like the RealNames system, this is hardly in line with the open environment of Berners-Lee's original vision of the web.

Other systems have been designed in an ad hoc fashion in order to archive a particular library's digital contents, but no the other resources on the web [12, 28]. Although fully operational in themselves, these systems are isolated from one another and from the public at large because they are not part of the web's architecture.

2.4. The Need for a New Approach

The existing solutions described here are all isolated, independent approaches that do not address the cause of the web's architectural flaw and do not sufficiently integrate with the web's existing architecture. Consequently, they cannot provide effective long-term solutions to the flaw.

We argue that it is the use of the URL and its reliance on the DNS that is the root of the problems identified here, because:

- the DNS is designed to map a hostname onto an IP address, whereas the web needs a system to map a resource name onto a location;
- the DNS deliberately constrains its namespace as it only has to deal with the names of servers, whereas the web needs an unconstrained namespace to cater for all different types of resources and the needs of their owners;
- neither the DNS nor the URL have any way of storing and referencing a resource's time of creation.

In order to fix the flaw the web needs a new service tailored to its own needs, which can provide referential integrity, an unconstrained namespace, and can locate a resource according to its position in time as well as space. In the next section, we present the design principles for such a service, which we have termed the *Resource Locator Service*.

3. The Resource Locator Service

3.1 Overview

The Resource Locator Service (RLS) has been designed as a name resolution service that is specific to the web. The service is fully backwards-compatible with the web's existing architecture, but provides new functionality that enhances it. It has been designed to work with the DNS as well as on its own, so that it does not have to completely replace the DNS in order for it to function effectively. As such, the RLS will only manage those resources that

are explicitly registered with it, giving the user the choice of whether they wish to use its advanced features or not, while maintaining full backwards-compatibility with the web's existing architecture. The service is designed to be deployed in an evolutionary way, becoming increasingly prevalent on the web until it eventually becomes the *de facto* name resolution service, leaving the DNS as the *Internet's* name resolution service.

3.2 Unconstrained Namespace

The RLS has been designed to accept any string as the name for a resource, allowing an infinite variety of naming schemes and namespaces to be used. As such, the RLS provides a technical solution to the constrained namespace problem, but does not define a way of avoiding namespace conflicts. However, this is a matter of policy rather than technology, and so will be left for future research.

We envisage the RLS being run along similar lines to the DNS, with individual nodes in the system being independently operated, but an organization such as ICANN managing the addition or removal of those nodes. However, unlike the DNS, no organization would have control over the namespace.

3.3 Transparent Resource Migration

The RLS helps to prevent link rot by providing a transparent resource migration service that maps a persistent name onto a dynamic location. Resources that wish to make use of the service must first register with it, after which they are free to migrate without breaking the links that reference them. Resources that do not register will still cause link rot should they migrate to a different location. As such, the RLS will not provide an immediate solution to link rot, but will act to retard its growth until all resources register with the service. Once this happens, link rot will only occur when a resource is no longer required and is deleted. However, we envisage third-party archiving services being employed to host such unwanted resources, with the RLS being used to maintain their persistent name. Although this does not guarantee the integrity of all links, we argue that it is the most appropriate level of integrity for the web, as persisting all resources forever would be impractical, and unwanted resources will not have many links referencing them anyway.

Resource migration enables the RLS to shift the responsibility for link management (and thus link rot) from the resource owner to an automated service. Because a registered resource's name persists across servers, the resource owner is not required to manage broken hyperlinks, as their integrity is guaranteed by the RLS. The owner must still inform the RLS of the new location, but we have automated the entire process through a new Resource Migration Protocol (RMP - see section 5.1), which remotely instructs the RLS to update the location of a migrated resource. To demonstrate this, our prototype includes a RMP client with a drag and drop interface for moving resources across servers using a style very similar to Microsoft's Windows Explorer (see section 7.1.3). This enables the resource owner to freely move registered resources across all web servers without any manual link management.

3.4 Temporal References

The RLS preserves the web's history through the use of a new *temporal reference*, which enables a resource to be identified according to its time of creation as well as its name. In this way, different versions of the same resource can share the same name, while still being differentiated by the RLS according to the reference's temporal attribute. This will also enable resources to be searched for according to their time of creation as well as their subject matter, enabling temporal search engines to be developed that show the state of the web and its resources at different points in time.

Temporal references require the resource and its content to be bound tightly together and treated as an atomic unit. Should the content need to change, a new resource must be created to contain it. However, rather than requiring a new name, the new resource can use the

existing name, but with a different temporal attribute. In this way, different versions of the same resource can be uniquely identified according to their time of creation, without requiring different names. References without a temporal attribute are assumed to be *current references*, which always reference the most recent version of the resource. This enables the temporal reference to resolve the semantic ambiguity of the hyperlink, as the current reference can be used to identify a web *page* whose content changes frequently, while a fully defined temporal reference can be used to identify a specific *version* of the web page according to the time that it was created.

Note that although the RLS binds a resource to its name and content in a way that appears similar to the specification of the URN, its functionality is very different. For example, the namespace of the RLS is completely unrestricted, whereas that of the URN is rigidly constrained. In addition, a URN can persist *beyond* the lifetime of its associated resource, whereas the RLS will persist a resource's name only as long as the resource exists.

4. Architecture of the Resource Locator Service

4.1. The Locator Network

The RLS is a distributed database, deployed across a network of nodes called *Locators*. A Locator is analogous to a Resolver [36], but is able to locate a resource through time as well as space by storing a resource's name, current location, and time of creation. This *Locator Network* performs a similar job to the DNS, but with granularity at the resource level.

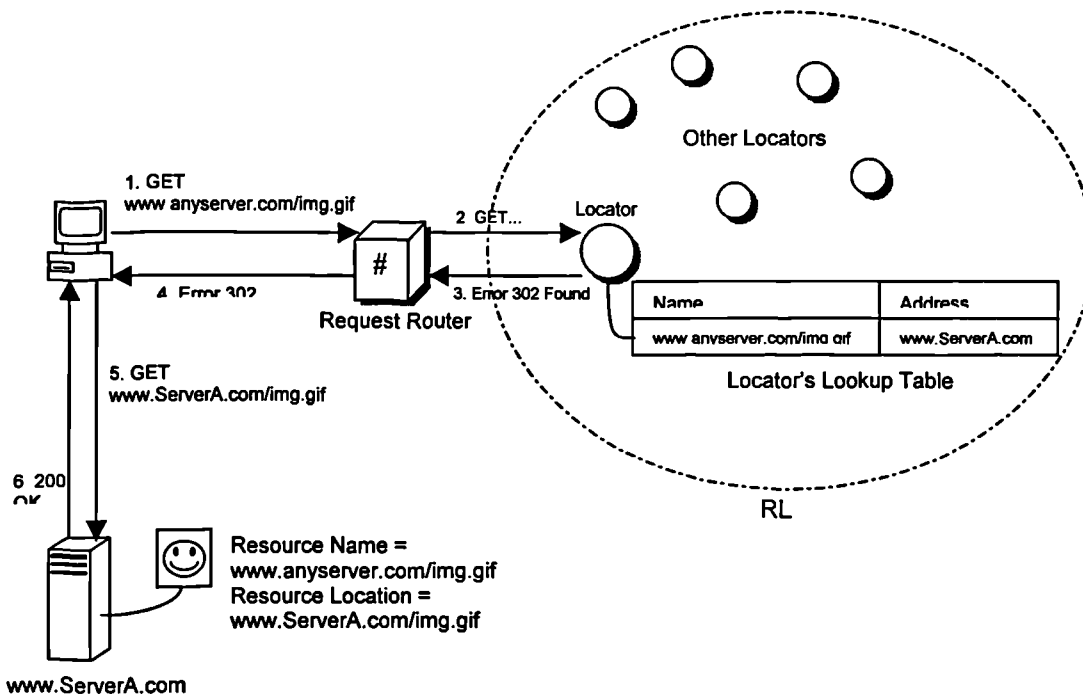


Figure 1 - Basic Architecture of the Resource Locator Service

Figure 1 shows a high level view of the RLS. The Locator network uses standard HTTP for communication, with the client being redirected to the correct location of the resource using the HTTP redirect mechanism [9]. Although this is not the most efficient approach, it facilitates backwards-compatibility, enabling all web entities to use the RLS.

The Locators take any string as the name of a resource and return that resource's location as a URL. The only constraint on the name used is that it must identify only one resource (although that resource may itself be replicated, and so have many values for its location in

the database). In this way, the RLS removes any technological barrier to future naming schemes, leaving policy alone to determine their structure.

4.1.1 Migration Approach

Of the five approaches described in section 2.1, the RLS uses the *name server* approach, as it is the most appropriate for the required functionality. Of the other approaches:

- the *lecture the user* approach advocates doing nothing as a solution, and so can be immediately discounted;
- *forward referencing* may provide referential integrity, but it still relies on the URL and thus the DNS's constrained namespace, and cannot support temporal references;
- the *search* approach will not scale to a system the size of the web [14], and the situation would be made worse if it had to manage temporal as well as spatial references;
- the *callback* approach provides referential integrity without providing a true naming service, and so the URL and the DNS would remain.

The name server approach, in contrast, is simply a distributed database, and so will scale to the size of the web and support temporal references. However, designing the system such that it interoperates with the web's existing architecture has been a major engineering challenge. Our solution to this problem is presented in the following sub-section.

4.2. Request Routing

Figure 1 is a high level representation of the RLS and depicts the client querying an appropriate Locator. However, in practice, a client must be made aware of exactly which Locator contains the required name/location mapping, but in a way that does not require the client or the hosting server to be altered. As such, some form of mediation is required between the client and the RLS that can transparently route the client's request to the appropriate Locator, without requiring any modifications in the client or server. This is difficult to achieve, however, as the constraints imposed on the RLS directly conflict with its distributed nature. For example, some distributed systems, such as the DNS or directory services, use the structure of the namespace itself to identify the correct node, but the RLS cannot, as the namespace must be left completely unconstrained. The alternative is a flat architectural configuration, with nodes arranged as peers and the namespace left unconstrained, but this requires the search approach to be used, which will not scale to a system the size of the web [14].

The IETF has also faced this problem with the design of the URN, and their proposal involves using the DNS to locate the correct node (termed a Resolver) in their Resolver Discovery Service [7]. A URN is sent to the DNS, which then forwards the URN onto the correct Resolver, which in turn resolves the URN to the correct URL. Essentially, the DNS's architecture is adapted so that it acts as an access system into a network of distributed name servers (the Resolvers), which perform the actual URN/URL mapping. Although this can work without requiring every browser and server to be changed, it has several disadvantages:

- It fundamentally changes the purpose of the DNS (from a name/address resolution service to an access service to another network).
- The DNS's importance in basic network routing prohibits its use in experimental work [3].
- URNs would still be constrained by the DNS namespace [7].

To resolve this problem for the RLS, we have developed a new system of mediation using an object called the *Request Router*, which transparently routes a standard HTTP request to the appropriate Locator in the RLS. The Request Router can work with any string as a resource identifier, and does not flood the entire RLS in order to locate the node with the required information. Furthermore, the system is fully backwards compatible with the web's existing architecture, and generic enough to provide mediation between the web and any distributed system.

4.2.2 *The Request Router*

The Request Router (RR) provides transparent, scalable mediation between the web and the RLS through the use of a *hash routing* algorithm. Specifically, a hash routing algorithm takes a string and maps it onto a hash space. The hash space is partitioned such that the string is mapped onto one and only one node in a distributed system [33, 40]. Using a hash routing algorithm as the basis for locating nodes in the RLS, therefore, enables any string to deterministically identify the Locator that contains the required name/location mapping. As the Locator is a database, it can be defined to store any type of information, and so the resource's location can be defined as any string. Thus, the hash routing algorithm solves the problem of how to use an unconstrained namespace for both a resource's name and location, whilst efficiently locating the correct Locator without flooding the system.

4.2.3 *The Hash Routing Algorithm*

The RR uses the same hash routing algorithm as the Cache Array Routing Protocol (CARP) [41], which maps a URL onto a specific cache in a distributed caching system, such that resources are distributed evenly across all caches in the system. The algorithm used in CARP works by mapping the URLs of resources that need to be cached onto a partitioned hash space, with each set in a partition being associated with one caching node [33]. The algorithm deterministically identifies the node as follows:

1. The URL of the resource is hashed.
2. The URLs of each of the caches in the array are also hashed in turn, with a weighting factor being applied that is set according to the physical characteristics of each node (see below).
3. The hash value of the resource and the hash values of the nodes are XORed together, producing a score for each resourceURL_hash/cacheNode_hash combination.
4. The cache node whose resourceURL_hash/cacheNode_hash combination scores highest is the one that hosts the resource.

Thus, given only the name of the resource and the names of all the machines in the array, the exact machine that holds the resource is uniquely and deterministically found. The resources are distributed uniformly across the system, but the weighting factor can be used to skew the distribution such that those nodes with a higher performance will host more of the resources.

4.2.3.1 *Adapting the Hash Routing Algorithm for the Resource Locator Service*

Although highly effective in large cache arrays, we have adapted the CARP protocol for use in the Request Router to better meet the needs of the RLS. Specifically, each node in a CARP system keeps a list of the URLs of all the caches in the system, and this causes a degree of network overhead. When applied to the RLS, however, every RR would need to know the URL of every Locator in the Locator Network, and periodically check for system configuration changes. This would create an unacceptable increase in network overhead, and would limit the types of device that could use the RR to those that could store and maintain the large lists of Locators that would be required.

The RLS avoids this limitation, however, by removing the weighting factor from the algorithm, and leaving the namespace of the resources open while restricting the namespace of the Locators. As such, a *URL pattern* is defined, which all Locators must use for their own name. The pattern encapsulates a number, which can be thought of as that Locator's identity number. Each number must be unique in the system, and all numbers must be sequentially ordered, starting from 0. An example URL pattern is:

http://www.nodeX.Locator.net/

where *X* represents the Locator's number in the system. For example, the first three nodes in the system (assuming a zero-indexing system) would have the following URLs:

http://www.node0.Locator.net/

http://www.node1.Locator.net/

http://www.node2.Locator.net/

Effectively, the URL pattern of the Locators acts as a *well-known* URL in a similar way to the well-known ports defined for TCP applications. Note that the URL pattern must be sequential, and there can be no gaps in the sequence. The URLs of a complete sequence of nodes, each of which has a URL that corresponds to the URL pattern, is therefore known as a *URL sequence*. In this way, the URLs of the Locators themselves become deterministic.

4.2.4 Updating the Request Router

Adding a new Locator to the RLS will cause the hash routing algorithm to implicitly re-map $1/n$ resources in an n -node system (note that n includes the new node) [33, 40]. As such, if a Request Router is unaware of this change in the system's configuration, then $1/n$ of its requests will go to the wrong Locator. However, the RR does not need to be synchronized with the configuration of the RLS, as the deterministic nature of the URL sequence enables it to detect any change automatically. Specifically, once the RR has the URL pattern for the RLS, it is a trivial matter for it to iterate along the resulting URL sequence, querying the existence of nodes at each point in the sequence. If a Locator fails to respond, then the RR has found the limit of the sequence. Thus, if a Locator cannot find a resource, the RR can simply query the existence of the Locators that have a node number that matches this limit (in case a Locator has been removed), or is one greater (in case a Locator has been added). If the limit remains unchanged, then the RR knows that the resource is unregistered with the RLS; otherwise, the RR simply rehashes the resource's name using the updated value, and sends the request to the newly calculated Locator. In this way, the RR is completely decoupled from the configuration of the Locator Network, and so any change in the configuration of the system does not result in a flood of update messages. Furthermore, the only information that the RR needs to store about the configuration of the system is the URL pattern and the number of nodes.

Note that the system's reliance on the URL sequence makes it vulnerable to node failure. Should a Locator fail, not only will its records not be available, but any RR that performs an automatic update during the failure will calculate the wrong number of nodes in the system, and will map most URLs onto the wrong Locator. However, the disruption can be limited if the RR continues to check for the existence of nodes beyond that at which no response is received, effectively enabling it to jump any holes in the URL sequence. Although the RR will still not be able to access the records in the failed Locator, it will at least know the correct configuration of the system, and so all other records will be available.

To further improve the resilience of the RLS, future work will look at introducing redundancy to the system, either by clustering several servers to provide a more fault-tolerant Locator design, or by using a *duplicated* hash routing algorithm, such as that proposed by [19]. Duplicated hash routing uses two hash routing functions and two cloned systems, one of which is a secondary system that acts as a backup in the event of a node in the primary system

failing. However, the benefits of this algorithm need to be determined, as although the reliability of the system is improved, the size and complexity are increased.

4.2.5 Integrating the Request Router into the Web

Because the RR is decoupled from the RLS, and needs only minimal information in order to function, it can be deployed virtually anywhere on the web. For example, it can be:

- *embedded into an HTML document as a Java applet, ActiveX control or even script.*
When the user clicks on a hyperlink, the click event can be captured by the embedded RR, the hash routing operation performed, and the location of the Locator discovered. Thus, the node location process occurs within the HTML page itself. This ensures total transparency and maximum backwards compatibility, but permits only HTML documents to use the RLS;
- *built into a browser.*
The browser automatically locates the appropriate Locator, allowing all servers to be unaware of the RLS, but requiring the client to be modified;
- *designed as a browser plug-in.*
The browser is *extended* rather than redesigned, with the RR being downloaded by the user when required. This provides seamless evolution, and a solution that is more backwards-compatible than the previous example. Again, all servers are unaware of the RLS;
- *built into a server, or added as a server module.*
The RR can be deployed on the server, which can perform the hash routing algorithm for each request it receives. This allows all browsers to be unaware of the RLS, and gives server owners the choice of whether to use the RLS or not;
- *embedded within a proxy server or a reverse proxy server.*
The proxy server intercepts the request, and routes it to the appropriate Locator. This requires reconfiguration rather than redevelopment, allowing all browsers, servers and resources to be unaware of the RLS;
- *designed into a layer 4 switch or policy based router.*
The switch or router contains the RR, transparently routing the request to the appropriate Locator without the knowledge of the client or the server. This provides total transparency and maximum backwards compatibility.

This allows the RR to be integrated into the web wherever it is required. In this way, the number of resources registered with the RLS can grow over time as more people decide they wish to use its services. As such, we envisage the adoption of the RLS to be evolutionary, rather than revolutionary, proceeding in a distributed way across different sectors of the web, as its services become useful to different types of user. For example, to begin with, small numbers of web authors may embed a RR within a HTML document. After a short period, server owners may decide to embed a RR into their servers in order to use the RLS without affecting the clients. From this, plug-ins can be made available for existing browsers, allowing resources to be located directly in the browser, via both the DNS and the RLS. Once a reasonable number of people use the RLS, Internet Service Providers can embed a RR into their proxy servers. Eventually, the RLS will reach a critical mass of users, whereby a RR will become an integral part of a browser and server, and thus part of the web itself. In this way, the RLS's database becomes populated over time by resource owners who choose to register their resources with it. As such, the database does not need to be initialized, and because it freely co-exists with the DNS, does not prevent non-registered resources from being accessed.

5. The Functionality of the Resource Locator Service

5.1 Transparent Resource Migration

In order to help solve link rot, the RLS must be able to dynamically reassign the location of a resource while persisting its name. This can be done using a new protocol that we have developed specifically for this task, called the Resource Migration Protocol (RMP), which is briefly described in this section (see figure 2). A more detailed discussion will be the focus of a future publication.

RMP is based on WebDAV (Web Distributed Authoring and Versioning [11]), a new extension to HTTP designed to enable group authoring of web resources. WebDAV has been chosen because of its new file manipulation semantics, such as the ability to lock files, which are implemented via HTTP. However, RMP is flexible enough to allow existing HTTP servers that are not WebDAV compliant to participate in the migration process, but without the extra safeguards that WebDAV provides.

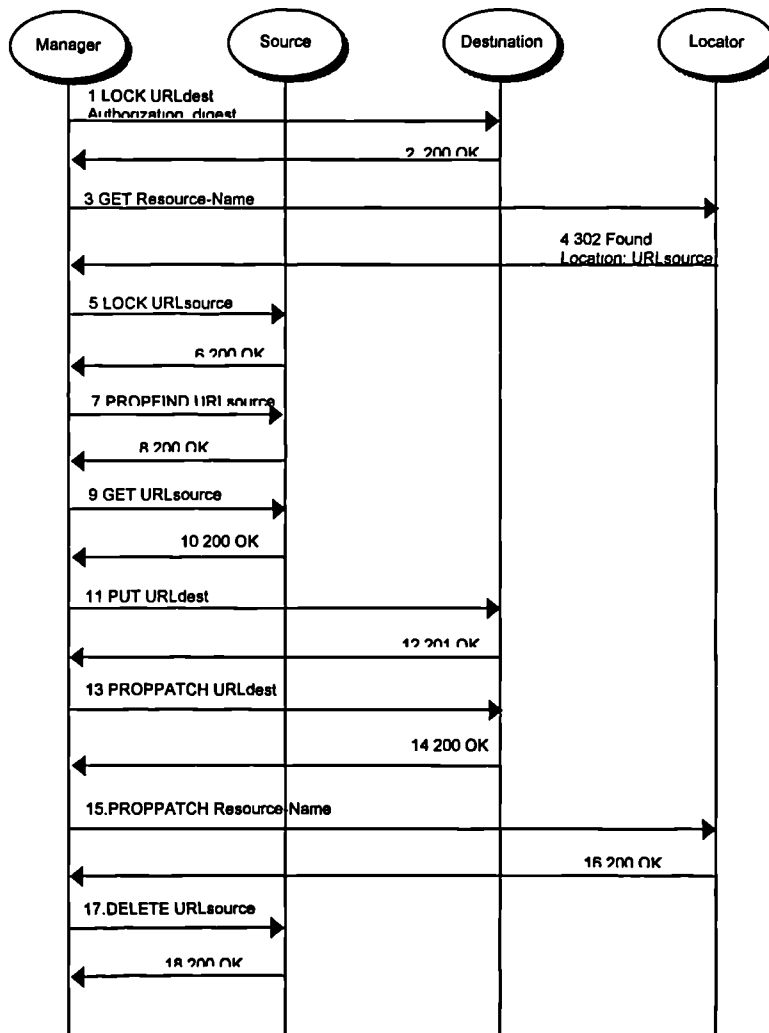


Figure 2 - The Resource Migration Protocol

5.1.1 The Resource Migration Protocol

The Migration Manager, defined as any entity that wishes to migrate a resource, manages the entire migration process, and is the only participant allowed to act as a client throughout, while the Source and Destination servers do not communicate with one another at all. The process begins with the Manager contacting the Destination in order to ascertain whether or not it is willing to host the resource. It does this by sending a WebDAV LOCK message (message 1) for the resource identified by URL_{dest} , together with authentication details (see figure 2). As the resource still exists on the Source server, there should be no resource physically located on the Destination that is bound to this location. Locking a null resource in this way has the effect of reserving the URL, ensuring that no other user can use URL_{dest} until the Manager unlocks the resource [11].

Upon a satisfactory response, the resource must be migrated from the Source to the Destination in such a way that it is accessible throughout the process. As such, the Manager must locate the Source using its own RR (messages 3 and 4), and LOCK the resource (messages 5 and 6), to ensure that it is not updated in the middle of the migration process. The Manager then sends the Source a WebDAV PROPFIND request (which allows a resource to be queried according to its attributes) in order to retrieve the resource's name, location and time of creation (messages 7 and 8), before sending a standard HTTP GET message to retrieve the resource itself (messages 9 and 10). The resource is copied to the Destination using a standard HTTP PUT message (messages 11 and 12), using URL_{dest} as the new location for the resource on the Destination. Note that although WebDAV provides MOVE and COPY methods for moving and copying a resource, they have not been defined for cross-server implementation; that is, a MOVE, for example, is only defined for moving a resource to a new location on the *same* server. This limitation prevents these methods from being used in the RMP, as WebDAV servers will not support moving a resource onto a different WebDAV server.

At this stage of the process, the resource is physically located on both the Source and the Destination. Before the resource on the Source is deleted, however, the Manager must update the appropriate Locator. It does this by sending a PROPPATCH message (message 15) to the Locator, which includes the resource's name and time of creation to identify the resource, and URL_{dest} as the property to be updated (i.e. its location). Once the Locator responds that the change has been successful, the resource located on the Source can be deleted using either a WebDAV DELETE message, or a standard HTTP DELETE request message. Once the DELETE response message (message 18) has been received, the migration process is complete.

5.2 Initializing and Updating the Resource Locator Service

The RLS is designed to be used in a way similar to that of the DNS; that is, a resource owner must first register the details of their resource in order to use the services of the RLS, and must then inform the RLS should the resource be deleted. The registration process acts to initialize the Locators' databases, and ensures that the resources that are managed are those whose owners explicitly requested the management service.

The Locators have been designed to enable a resource owner to automatically register and deregister a resource through the Locator's interface, using WebDAV messages. In order to add a resource's details to the RLS, a client can send a WebDAV PROPPATCH message [11] to the appropriate Locator, with the resource's name and time of creation properties encoded in the message body. The WebDAV specification defines PROPPATCH to set and remove properties, but the Locator cannot allow it to change the name of a resource or its time of creation (see section 3.2). As such, the Locator's interface restricts the client to adding or removing entire records only, with RMP used to automatically modify the location data. In this way, the complete functionality of the RLS is fully automated, while referential integrity is enforced.

In order to remove a resource's details from a Locator, the client must send it a PROPPATCH message containing a *remove* XML element (see [11], section 12.13.1), which acts to delete the details from the locator, but not the resource from the web. The Locator essentially delegates that responsibility to the resource owner, viewing the message as a request for the resource to leave the RLS, rather than for the resource to be destroyed. As such, the Locator will include the current location of the resource in a *location* header in the response message, allowing the client to send a HTTP DELETE message to the hosting server to physically delete the resource, if required.

Note that new HTTP headers could also achieve the same purpose as PROPPATCH, but the use of WebDAV messages is more consistent with RMP, and the semantics of the messages fit well with the needs of the Locator. Specifically, PROPPATCH "...processes instructions specified in the request body to set and/or remove properties defined on the resource identified by the Resource-URI" [11]. In this way, the Locator acts as a third party that manages the properties of the resource on behalf of its current host. In addition, the WebDAV error message *409 conflict* is used if a client tries to change the name of a resource within the Locator, as this message informs the client that it has "...provided a value whose semantics are not appropriate for this property" [11].

5.3 Temporal References

5.3.1 Defining the Temporal Reference

A client must be able to query a Locator for a resource according to its time of creation, and so some form of temporal identifier is required. As such, although the RLS can use any string as an identifier, we have specified two types of temporal reference in order that temporal referencing can be used immediately. Specifically, we have designed the Locators to work with standard URLs with a *timecreated* temporal component appended as a Query String (e.g. <http://www.asever.com/index.htm?timecreated=Sun,%2006%20Nov%201994>), which allows existing URLs to be used as temporal references; and with a new temporal URL scheme that we have defined as a more long term, architectural solution. The new temporal URL scheme conforms to the encoding rules defined in [4], and encapsulates the same semantics of the URL, but with the addition of a temporal component. Specifically, the new scheme, called TURL (Temporal Uniform Resource Locator), has been defined as:

turl://authority/path;time-created?query

The *authority* component of the TURL is identical to that of the URL (i.e. the domain name of the hosting server). The *path* component, too, is identical to the URL, but with one exception: a semi-colon separates the path that the server uses to locate the resource from the temporal information used to identify the time that the resource was created. The *query* component remains as it is defined for the URL, but the whitespace of the temporal component has been replaced with a dash (-) for clarity. Thus the URL:

<http://www.asever.com/index.htm?timecreated=Sun,%2006%20Nov%201994>

can be re-written as a TURL as:

turl://www.asever.com/index.htm;Sun,06-Nov-1994

In addition, as HTTP essentially forms the interface between the RR and the Locator, we have had to extend it in order to map the temporal component of the TURL onto a HTTP header. HTTP's existing headers already encode temporal information, but they are largely used for caching, and are normally sent by the server rather than the client. For example, the *Last-modified* entity header is used to represent the time at which the resource was last modified, which is another way of saying the time at which the resource was created.

However, it can only be used by servers in a response message, and cannot be used by a client at all. Equally, the *Age* entity header [9], which provides the estimated age of the resource on the origin server, is also a response header, only sent by a server (usually a caching proxy server). Alternatively, the *Date* header field is a general header, which can be used by both client and server, but only to represent the date and time at which the *message* was originated, not the resource [9]. Finally, the ETag header could encode the temporal information, as it provides a means of encoding user-defined values, but it, too, is a response header [9].

As such, faced with the decision of subtly altering the semantics of HTTP or defining a new general header, we have chosen the latter option, and defined a header called *time-created*, which can be used by both client and server, and which defines the time at which the resource was created. The value of the new header must be formatted according to [5], and it must map exactly onto the temporal component of the TURL. The new header provides the preferred means for querying a Locator according to a resource's time of creation, thus separating the temporal information from the resource's name. In this way, any appropriately specified namespace is able to become a temporal reference, enabling the RLS to retain its unconstrained namespace.

5.3.2 Defining the Scope of the Temporal Reference

A temporal reference supported by the RLS can enable one resource to persist across time, but not the resources behind any hyperlinks that might be embedded within it. For example, a HTML document registered with the RLS may contain several hyperlinks, but if the resources underlying the hyperlinks are not registered with the RLS, then they may not persist. As such, the RLS can only prevent link rot for those resources that it has been instructed to manage, and so web-wide link rot prevention can only be achieved if the RLS manages all web resources.

In addition, transient resources, such as dynamically created HTML documents, or streaming audio or video, are also not covered by the current design of the RLS and the temporal reference. This is because the TURL simply extends the existing URL protocol to encompass time, rather than adding any new functionality, and an existing URL references the object that creates a dynamic resource or a multimedia stream, rather than the transient resource itself. For example, a URL might identify an application behind a CGI (Common Gateway Interface) gateway, which in response returns a dynamically generated HTML document, but it does not identify the HTML document. Similarly, temporal references may enable the application to persist (although their definition does not cover persisting the application's state, merely its existence as a discrete file), but they do not cover its output, unless it is explicitly saved as a permanent web resource and given its own (temporal) URL.

6. Changing the Configuration of the Resource Locator Service

When the configuration of the RLS changes, the hash routing algorithm will re-map $1/n$ (where n = total number of Locators) of all records onto a different Locator. This will make the RR incorrectly route $1/n$ of all subsequent requests, unless the appropriate records are transparently migrated to the correct Locator. As such, the RLS must carefully manage *transparent record migration* (termed to reflect the fact that it is individual records in a Locator's database that must migrate) if it is to remain robust in the face of a changing configuration.

To manage this migration, we have developed the Locator Control Protocol (LCP), which allows all Locators in the RLS to be controlled such that the location of remapped records can be corrected without the knowledge of the RR. LCP is based on HTTP, ensuring its compatibility with existing web server technology. However, its full specification is beyond the scope of this paper, and so the following sections will present an overview of the protocol only. Section 7 will discuss the performance implications of the protocol.

The role of the LCP is to ensure that a Locator can be added to or removed from the RLS transparently, such that a RR is able to access all records throughout the system's

configuration change. The key to achieving this is to enable both configurations to co-exist for a short period by copying those records that must move, before the existing configuration is deleted to make way for the new one. In this way, all records are accessible whichever configuration the RR attempts to use.

6.1 Adding a New Locator

When a Locator is added to the system, a RR will only notice the change when it updates itself and the new Locator has adopted a domain name that complies with the appropriate URL pattern. In this way, the adoption (or removal) of a RLS-compliant domain name acts as a switch: with the domain name, a Locator is recognized by a RR as part of the RLS; without it, the Locator is not recognized, and so will simply be ignored. As such, by first copying all migrating records to their new locations *before* the new Locator adopts its new domain name (figure 3a), the LCP can enable both configurations to co-exist, ensuring that all records are accessible both before and after the new Locator is recognized by the RR.

The protocol requires the new Locator to act as the record migration manager, coordinating the migration process to ensure integrity of the records. While the migration is occurring, all Locators can still perform their standard name resolution service. Once the new Locator adopts a domain name that complies with the URL pattern, both configurations effectively co-exist. Those RRs that have not updated will be able to access the records in their existing location; those RRs that have updated, will be able to access the records at their new location (figure 3b). Once in this state, the old configuration can safely be deleted (figure 3c), causing those RRs that have not updated to receive an Error 404 when they try to access a remapped record, which will prompt them to update and thus to recognize the new configuration. In this way, no configuration updates need be sent to any RR throughout the entire process.

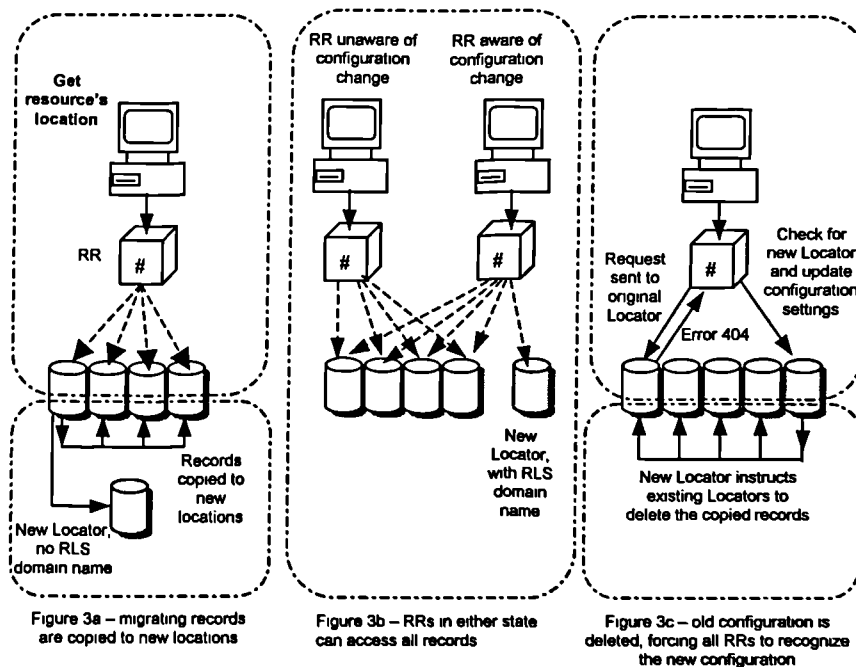


Figure 3a-c – Adding a New Locator

6.2 Removing an Existing Locator

Removing a Locator requires a different approach, however, as deleting its RLS-compliant domain name may leave a hole in the sequential numbering of the URL sequence, confusing the RRs. The process begins when the detaching Locator ($Locator_{detach}$ - coloured black in figure 4), acting as a coordinator, instructs all other Locators in the system that the configuration is about to change, thus causing them to copy the records that must migrate to their new locations. Note that some of the records will be copied onto $Locator_{detach}$, even though it is about to leave the system (figure 4a). Once this is complete, $Locator_{detach}$ remains in the RLS, and instructs the *last* Locator in the sequence ($Locator_{last}$ - coloured grey in figure 4) to detach itself from the system, even though $Locator_{last}$ is not the one that wishes to leave (figure 4b). In this way, the RLS shifts to the new configuration, with the existing configuration still operational (note that the RRs that have not updated may attempt to reach $Locator_{last}$, but will not receive a response; this will cause them to update automatically, however, thus moving them to the new state).

Once the new configuration has been reached, $Locator_{detach}$ will instruct the (now removed) $Locator_{last}$ to delete all of its records, before copying its own records over to make both Locators mirrors of one another. In addition, $Locator_{last}$ will also be given the same domain name as $Locator_{detach}$, making the two Locators identical clones (figure 4c). Once this happens, $Locator_{detach}$ is free to detach itself by removing its IP address from the DNS entry for its domain name, leaving the RLS in the new configuration. Again, the process of removing a Locator requires no synchronization messages from any Locator, as all RRs will automatically update themselves.

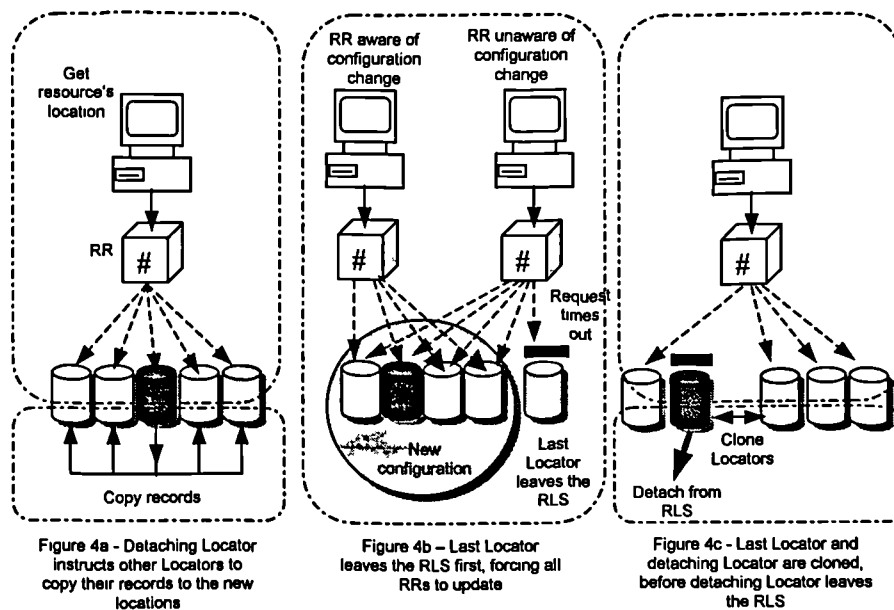


Figure 4a-c – Removing an Existing Locator

7. System Evaluation

To test the concepts discussed here, we have developed a prototype RLS, which comprises:

- a small network of Locators;
- a Request Router;
- a HTTP proxy server;
- a management interface.

This section presents the implementation details of the prototype, and performance data that we have gathered to demonstrate the scalability of the design.

7.1. Prototype Implementation

7.1.1 Prototype Locator

The Locator has been designed as a web server using Microsoft IIS on NT 4 Server, which uses Active Server Pages (ASP) to implement the functionality, and integrates with a Microsoft Access database that stores the name, location, and time of creation of each managed resource. The same resource name can reference multiple entries in the database, as each resource may have multiple locations (i.e. replicated resources) and multiple times of creation (i.e. when its content is changed). As such, the resource's name, time of creation, and location represent a compound key that together uniquely identify a single record, allowing the Locator to support replication and temporal referencing.

When a Locator receives a standard HTTP request, it looks up the resource's details in its database. If it contains the resource's name/location mapping, it returns a *302 Found* HTTP response message, with the current location of the resource contained within its *location* header; otherwise it returns a *404 Not Found* HTTP error message. In this way, a client can communicate with the RLS transparently, providing full backwards compatibility. If the Locator receives a HTTP request with a 'HEAD' method, it will simply return a *HTTP 200 OK* response, enabling RRs to safely query for the existence of a Locator.

Finally, the Locator supports the RMP, using both WebDAV-enabled and standard HTTP servers as source and destination machines, enabling transparent resource migration to be implemented across all web servers.

7.1.2 Prototype Request Router

The Request Router is perhaps the most important part of the system, as it must be integrated into the web's existing architecture. To do this, we have created a Request Router object in C++ and embedded it into a simple HTTP proxy server, which routes the incoming request to the appropriate Locator, and then downloads the resource (if found) from the appropriate server. Any user who wishes to use the RLS can configure their browser to use the proxy server, enabling all legacy browsers and servers to use the RLS transparently. The proxy server application is also small enough for deployment on a client machine, allowing it and the browser to co-exist on the same machine if required. Future versions of the RR will include an ActiveX version, allowing the RR to be embedded into HTML pages, browsers such as Microsoft's Internet Explorer, or standard web servers.

The RR's client-side interface has two functions that are used to identify the correct Locator. The first, *RouteRequest()*, takes the name of a resource, and returns the appropriate Locator's URL with the resource's name appended onto it as a Query String (e.g. *http://www.node1.Locator.net/query?resourcename=http://www.asever.com/aresource.htm*). This URL can then be sent directly to the appropriate Locator without the need for adding any new HTTP headers. The second, *GetLocatorByName()*, returns the appropriate Locator's URL *without* the resource's name being appended as a Query String. The resource's name must be encoded in a HTTP request header in a subsequent HTTP GET message. In both functions, the location of the resource is provided by the appropriate Locator via a HTTP redirect message (*302 Found*).

The RR also has functions that enable the URL pattern to be changed (thus allowing it to interface with other distributed systems on the web), and a function called *Update()*, which enables it to determine the number of Locators in a network by performing an automatic update.

7.1.3 A Prototype Management Interface and Resource Migration Protocol Client

In order to test the system, we have developed a management interface for controlling the Locators (see figure 5). The interface includes a RMP client, which enables it to act as the migration manager during a resource migration operation, and a Request Router object, enabling it to query the Locator Network.

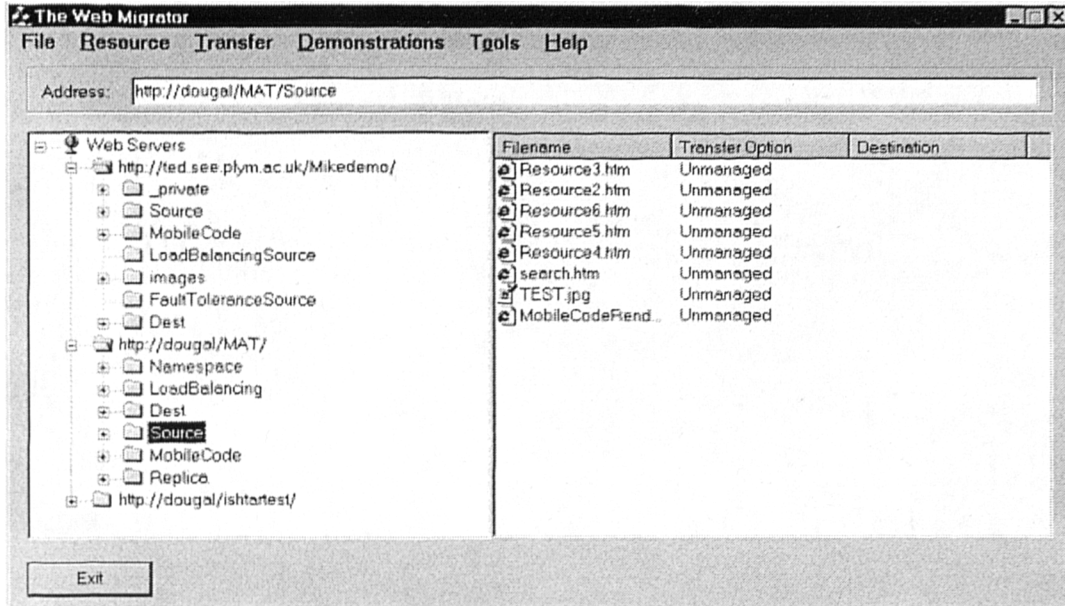


Figure 5 - Prototype Management Interface

The user interface is similar to Microsoft's Windows Explorer, allowing the user to drag and drop resources across web servers, while the underlying RMP functionality automatically updates the appropriate Locator's name and location details. The interface demonstrates the backwards-compatibility of the system, as the web servers shown in figure 5 have not been altered in any way, and are completely unaware of the RLS as a system.

7.1.4 Enhanced Web Services

In addition to the prototype RLS, we have developed several extra services that build on the services provided by RLS to extend the functionality of the web. These services take advantage of the RLS's resource migration mechanism, in order to provide transparent fault tolerance, load balancing, and mobile code functionality to existing web servers. Due to space restrictions, however, these services will not be discussed here, but will instead be presented in a future paper.

7.2 Scalability and Performance Issues

The prototype contains code that instruments performance, allowing us to measure its impact on standard web browsing. The results from our measurements, together with a general discussion on the scalability of the design, are presented in this section.

7.2.1 Network Overhead

Hash routing is a very fast algorithm for locating a node in a distributed system, providing a deterministic request resolution path through an array of machines, which results in locating a specific node in a single hop [24]. As such, the network overhead introduced by the RLS for both a successful and an unsuccessful resolution operation is *always* two additional HTTP messages (either a GET and an *Error 302 Found* response, or a GET and an *Error 404 Not Found* response).

If the RLS cannot find the resource, then a client application may contact the DNS if required, and if this is successful, the round-trip time to the RLS via the RR has been wasted. If, however, the RLS is completely integrated into the web, such that the DNS is not used to find the locations of resources, then all resources will be registered, and an *Error 404* means that the resource does not exist on the web, not just in the RLS. As such, there will be no added overhead, as the resource is unattainable.

Note that the RLS will not attempt to contact the DNS to find a resource, as it may not be appropriate in all cases. For example, a server hosting a RR may have registered all of its resources with the RLS, and so an *Error 404* means that the resource does not exist, not just that it is not registered. As such, it would serve no purpose for the RLS to contact the DNS in this situation, and so the RLS only manages its own *registered* resources, leaving clients to determine what to do with those that are unregistered.

7.2.2 CPU Overhead

The design of the RLS is such that the network overhead is constant, regardless of how many Locators are in the system, whereas the CPU overhead required by the RR scales linearly with respect to the number of Locators. As such, the scalability of the design is constrained more by CPU overhead than network overhead.

The linear scaling of the RR results from the hash function being used to distribute a set of records across many Locators, rather than to generate a unique value each time it is used, and so it does not have to worry about managing collisions, as the same result (i.e. the identified Locator) can be used many times for different resource names. The function distributes the records by hashing the URL of each Locator in the system, and as the time taken to hash each URL is virtually uniform (dependent solely upon the number of characters in the URLs that are hashed), the CPU overhead increases linearly with respect to the number of URLs (and thus Locators) it must hash.

We tested our Request Router on a Pentium Pro 200MHz with 64MB RAM, a Pentium III 400MHz with 128MB RAM, and an Athlon 1.1GHz with 128MB RAM. We set the number of nodes that the RR believed existed within the RLS to different levels, and measured the length of time that the RR took to identify the correct Locator. The results are presented in figure 6, which clearly reveals the linear relationship between time and the number of Locators. The results show that for small numbers of Locators, the time taken is insignificant, and that even with more Locators, the time taken is still small. As such, even with a relatively slow machine such as the Pentium Pro 200MHz, the RR can determine the correct Locator from a 10,000-node Locator Network in only 0.35 seconds.

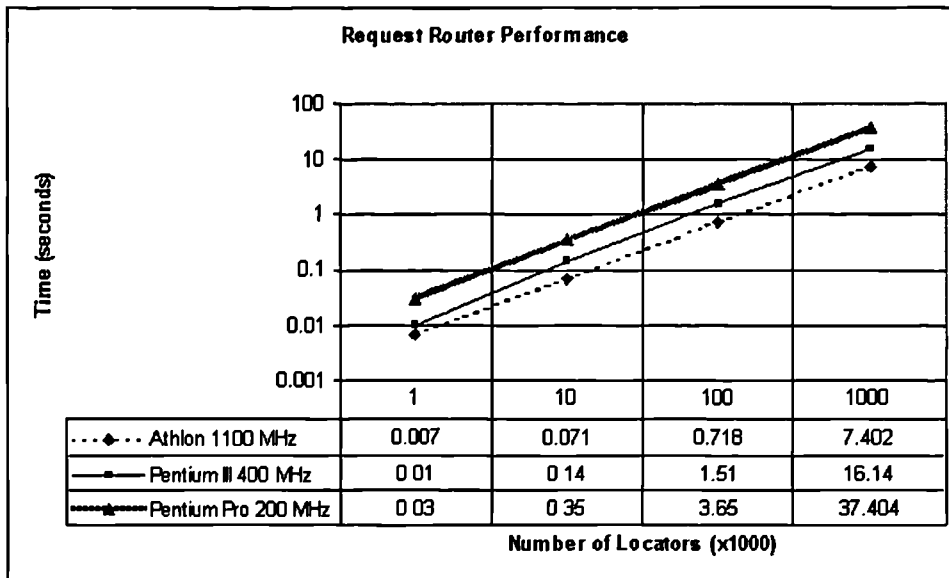


Figure 6 - Performance of the Request Router

In addition, the prototype RR was designed for experimental purposes, and is non-optimal. Specifically, it refreshes every Locator URL for every request that it routes, but unless the number of Locators changes, these hash values will remain static. As such, a more optimal design would cache the hash values in memory, and only rehash them when the configuration changes, thereby drastically reducing the length of time it would take to locate a Locator.

7.2.3 Total System Overhead

The total overhead introduced by the system was measured to provide a real-world indication of the system's performance. To do this, we first measured the time it took to visit the homepage of *www.lycos.co.uk* using a standard browser (Microsoft IE 5.0) and no RLS. The web site was visited 25 times, with the browser's cache deleted each time. We then connected the browser to our proxy server with the embedded RR, and visited the same sites again, once more taking 25 distinct measurements. The experiment was run using an Athlon 1.1 GHz PC with 128MB RAM, which acted as the proxy server with an embedded RR, and a Pentium Pro 200MHz PC with 64MB RAM, which encoded the functionality of the Locator. Both machines used Microsoft Windows NT 4 Server, and were connected via a 10Mbps Ethernet LAN.

The RR in the proxy server was manually configurable, allowing us to set the number of Locators according to requirements. For this experiment, we varied the number of Locators in the system from one to 1 million. However, to avoid having to physically deploy 1 million Locators, we reconfigured the proxy server so that it always forwarded the request onto the same Locator, regardless of which one the RR actually identified. The Locator would then return an *Error 404* message, which would cause the proxy server to redirect the request to the origin server, from where the resource would ultimately be retrieved. Because the overhead for the RLS is the same whether the resource is found or not (i.e. one extra HTTP request and one extra HTTP response), the measurements of the overall system overhead remained unaffected. In addition, this configuration removed any differences between servers that would have been introduced had each Locator been deployed on a separate physical machine.

The results presented in table 1 show the time taken to visit the Lycos web site both without the RLS, and with it, using one, 1,000, 10,000, 100,000, and 1 million Locators in the system. Each value is the 10% trimmed mean of 25 trials, with the overhead calculated by subtracting the mean from the value obtained without the RLS. The results show that the

overhead introduced by the RLS ranges from 0.869 seconds with only one Locator in the system, to 8.38 seconds with 1 million Locators. However, despite the large overhead for 1 million Locators, it remains small up to 100,000 Locators, with 1.582 seconds recorded.

Number of Locators	Download time for www.lycos.co.uk (time without RLS = 7.608 sec)	Overhead
1	8.477 seconds	0.869 seconds
1,000	8.483 seconds	0.875 seconds
10,000	8.546 seconds	0.938 seconds
100,000	9.190 seconds	1.582 seconds
1,000,000	15.985 seconds	8.377 seconds

Table 1 - Overhead introduced by the RLS with different configurations

The results show that the RLS introduces negligible overhead for a configuration of 10,000 Locators or less, and a relatively small overhead up to 100,000. However, it should be noted that neither the design of the RR or the proxy server are optimal, and that significant performance improvements can easily be made. We expect such improvements to enable the deployment of a 100,000 Locator system with negligible overhead.

7.2.4 Scalability of the Overall Design

In terms of growth, the hash routing algorithm can scale to over 4 billion ($2^{32} = 4,294,967,296$) Locators, performing single-hop resolution throughout [24]. Assuming each Locator can store the names and locations of 1 million resources (which, assuming an average URL of 50 characters, will require a database only 50 MB in size), today's web, with over 1 billion documents [15], would need the deployment of 1,000 Locators to fully manage all resources, which will take the RR on a Pentium Pro 200 machine just 0.03 seconds. However, even 10,000 Locators will only take 0.35 seconds, and this can accommodate ten times as many resources as currently exist on the web. Clearly, the configuration of the RLS can be better optimized, but this provides a good indication of the scalability and practicality of the design.

7.2.5 The Cost of Changing the Configuration

Changing the configuration of the RLS is not a time-critical process, as the name resolution service provided by the RLS is unaffected throughout the change. However, the change should still occur in a reasonable time-frame, and with a reasonable amount of network traffic, and so this section presents an estimate of the order of time that will be needed for a new Locator to be added to the system. Note that no estimations are provided for removing a Locator, as the operations are very similar, and so the time taken will be of a similar order.

The addition of a new Locator involves two steps that could significantly affect the time taken to update the system:

- determining which records need to move;
- physically moving the records.

The other steps involve data manipulation, such as deleting records, which will not negatively affect the scalability of the design or the time taken to change the configuration, and so will not be considered in the following calculations.

The first step involves every record in the system being processed by a RR whose node configuration is set at one node higher than its current value (i.e. set to $n + 1$). The time taken to do this can be significantly reduced if each Locator works in parallel with its peers, processing only the records contained in its own database. This is how the LCP operates. As such, ignoring the network overhead of the LCP, the time taken for step one will be:

$$\frac{Rt_r}{n+1}$$

where R is the total number of records in the system, and t_r is the time taken to process one record. Clearly, for the same R , the time will decrease with the number of Locators in the system, and as section 7.2.2 has shown, t_r is very small on even a low powered machine. Thus, the cost incurred by this first step is small. For example, suppose the existing configuration has 999 Locators managing 10^9 records, with an Athlon 1100MHz processor inside each Locator, giving $t_r = 0.007$ seconds (see figure 6). In this scenario, the time taken for step 1 will be:

$$\frac{10^9 \times 0.007}{999 + 1} = 7000 \text{ seconds (or 1 hour 57 minutes, 7 seconds).}$$

The cost incurred by the second step is dependent upon the number of Locators in the system and the number of records. When a new Locator is added to an n -node system, the records that are re-mapped will be evenly distributed across all Locators in the RLS [40]. If R is large compared to n , then every Locator in the system will contain records that are re-mapped. As such, each Locator will evenly distribute $1/(n+1)$ of its own records to the n other Locators in the RLS. This results in n Locators transmitting to n Locators (including the newly added one), resulting in the propagation of up to n^2 messages. This value represents the message limit, however, as each message can carry more than one record if required.

The time taken to transmit these messages can be shown to be acceptable for even the worst-case scenario, in which the configuration results in the maximum number of messages being sent (n^2), and only one message is in transit at any one time. For example, the configuration of step one is such that the addition of a new Locator requires the maximum number of messages to be sent for this configuration (i.e. $n^2 = 1,000 \times 1,000 = 10^6$ messages). The time taken to send these messages can be estimated if the data transfer rate between Locators is known, as well as the number of bytes in each record. As such, if we assume that a conservative data transfer rate of 1.544 Mbps can be maintained between Locators, and the average record size is 150 bytes (50 bytes each for name, location, and time of creation), then the total time taken to transmit all messages (ignoring protocol overhead and converting bytes to bits) is:

$$\frac{1,000,000 \times 150 \times 8}{1,544,000} = 777.20 \text{ seconds (or 12 minutes 57 seconds).}$$

Thus, for this scenario, the *total* time taken to complete the addition of a new Locator is only 2 hours, 10 minutes, 4 seconds, which is entirely acceptable. Furthermore, this figure represents a maximum value, and can be decreased by sending messages in parallel, and by balancing the number of records per Locator with the number of Locators in the system, according to the available bandwidth and the performance requirements of the RRs. Finally, it is worth reiterating that a configuration change is not a time-dependent task, as the RLS is fully operational throughout the change, and it will not happen often, as the configuration of

the RLS should remain stable for relatively long periods of time. As such, the design of the RLS remains scalable and practical for a system the size of the web.

8. Discussion and Conclusions

Security is critical within the RLS, and is an area that needs further work before the RLS can be deployed. The RLS should use the HTTP digest authentication scheme [9], as it is stronger than the basic scheme, but both schemes have been determined too weak for the WebDAV working group, which faces similar problems, and is working on its own solution [10]. The RLS must use strong authentication techniques, as malicious use could potentially route requests to unwanted resources. As such, we anticipate future versions of the RLS to use the WebDAV Access Control Protocol [34] for its authentication requirements.

In addition, our prototype system has so far only been tested locally as a proof of concept. The next step is to build a bigger system and to stress test it under varying loads. However, its design is based on an existing, commercially proven distributed system (CARP), and so we expect it to stand up well to such a test. The LCP will also be redesigned to provide a more optimal solution that employs concurrent operation.

Overall, the Resource Locator Service provides an effective and elegant approach that addresses the problems of link rot, a shrinking namespace and a lost history, by providing a replacement for the URL and the DNS. We believe the RLS offers a better solution than existing systems, as it offers a single, coherent, architectural solution, which addresses all three problems at once. Further, the RLS is fully backwards-compatible with the web's existing architecture, yet extensible enough for it to be future proof. A web page creator, a browser developer, or a proxy server developer, can use the RLS today without affecting any other system within the web. Equally, the system should be scalable enough for it to become an integral part of the web's architecture in future years.

The resource migration aspect of the RLS will also enable it to become part of a nascent platform for distributed computing on the web, providing services such as fault tolerance and load balancing, and allowing new services to be deployed, including a temporal search engine, that are currently impossible to implement on a web-wide scale. We intend to develop these services to fully exploit the potential of the RLS in future work.

Acknowledgements

We would like to thank Paul Dowland for his help with the experiments that were conducted for this paper.

References

- [1] Babich, A, Davis, J, Henderson, R, Lowry, D, Reddy, S and Reddy, S, DAV Searching and Locating, Internet Draft, <http://www.webdav.org/dasl/protocol/draft-davis-dasl-protocol-00.html>, April 20, 2000.
- [2] Berners-Lee, T, Cool URIs Don't Change, 1998, <http://www.w3.org/Provider/Style/URI>.
- [3] Berners-Lee, T and Fischetti, M, Weaving the Web – the Past, Present and Future of the World Wide Web by its Inventor, Orion Business Books, 1999, p. 133.
- [4] Berners-Lee, T., R. Fielding and L. Masinter (1998), "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, <http://www.ietf.org/rfc/rfc2396.txt>
- [5] Braden, R, Requirements for Internet Hosts – Communication layers, STD 3, RFC 1123, October 1989.
- [6] Briscoe, RJ, Distributed Objects on the Web, BT Technology Journal vol. 15 No 2, April 1997, pp.157-171.
- [7] Daniel, R and Mealling, M, Resolution of Uniform Resource Identifiers using the Domain Name System, RFC 2168, June 1997.

- [8] Evans, MP, Phippen, AD, Mueller, G, Furnell, SM, Sanders, PW and Reynolds, PL, Strategies for Content Migration on the World Wide Web, Internet Research, vol. 9, no. 1, 1999, pp25-34.
- [9] Fielding, R, Irvine, UC, Gettys, J, Mogul, JC, Frystyk, H, Masinter, L, Leach, P, Berners-Lee, T, HyperText Transfer Protocol – HTTP/1.1, RFC 2616, June 1999.
- [10] Franks, J, Hallam-Baker, P, Hostetler, J, Lawrence, S, Leach, P, Luotonen, A, Stewart, L, HTTP Authentication: Basic and Digest Access Authentication, RFC 2617, June 1999.
- [11] Goland, Y, Whitehead, J, Faizi, A, Carter, S and Jensen, D, HTTP Extensions for Distributed Authoring – WebDAV, RFC 2518, February 1999, <http://andrew2.andrew.cmu.edu/rfc/rfc2518.html>.
- [12] Hogarth, J, The National Register of Archives/ARCHON: a study to inform a development strategy for a National Name Authority File, for the Historical Manuscripts Commission, September 1999, <http://www2.hmc.gov.uk/pubs/jhreport.htm>.
- [13] ICANN Announcement, November 16th 2000, <http://www.icann.org/announcements/icann-pr16nov00.htm>
- [14] Ingham, D, Caughey, S and Little, M, Fixing the ‘Broken-Link’ Problem: The W3Objects Approach, in: The Fifth International World Wide Web Conference, Paris, France, May 6-10 1996.
- [15] Inktomi and NEC Research institute, <http://www.inktomi.com/webmap/>.
- [16] Internet Corporation for Assigned Names and Numbers, Uniform Domain-Name Dispute-Resolution Policy, <http://www.icann.org/udrp/udrp/-policy-24oct99.htm>, 24 October 1999.
- [17] Kahle, B, Preserving the Internet, Scientific American, March 1997. See also the Internet Archive, www.archive.org.
- [18] Kappe, F. A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems, in *J.UCS* Vol. 1, No. 2, Springer, February 1995, pp. 84-104.
- [19] Kawai, E, Osuga, K, Chinien, K and Yamaguchi, S, Duplicated Hash Routing: A Robust Algorithm for a Distributed WWW Cache System, in: IEICE Trans. Inf. & Syst., Vol.E83-D, No.5, May 2000.
- [20] Kosters, M, Massive Scale Name Management: Lessons Learned From the .COM Namespace, in: The Workshop on Internet-scale Software Technologies, University of California, Irvine, California, USA, August 19-20, 1999, <http://www.ics.uci.edu/IRUS/twist/twist99/>.
- [21] Lawrence, S and Lee Giles, C, Accessibility of Information on the Web, Nature, Vol.400, 8 July 1999, pp107-109.
- [22] Lyman, P and Kahle, B, Archiving Digital Cultural Artifacts, Dlib Magazine, July/august 1998, <http://www.dlib.org/dlib/july98/07lyman.html>.
- [23] Mealling, M, Requirements for Human Friendly Identifiers, Internet Draft, June 1998, <http://www.ics.uci.edu/pub/ietf/uri/draft-mealling-human-friednly-identifier-req-00.txt>.
- [24] Microsoft Corporation, Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0, 1997, <http://msdn.microsoft.com/library/backgrnd/html/carp.htm>.
- [25] Mockapetris, P, Domain names – concepts and facilities, RFC1034, November 1987, <http://www.ietf.org/rfc/rfc1034.txt>.
- [26] Mockapetris, P, Domain names – implementation and specification, RFC1035, November 1987, <http://www.ietf.org/rfc/rfc1035.txt>.
- [27] Moore, K, Location-Independent URLs or URNs Considered Harmful, Internet Draft, 7 January 1996, <ftp://cs.utk.edu/pub/moore/draft-ietf-uri-urns-harmful-00.txt>.
- [28] Moore, R, Baru, C, Rajasekar, A, Ludescher, B, Marciano, R, Wan, M, Schroeder, W and Gupta, a, Collection-Based Persistent Digital Archives – Part 1, D-Lib Magazine, Volume 6 Number 3, <http://www.dlib.org/dlib/march00/moore/03moore-pt1.html>, March 2000.
- [29] NetCraft WebServer Survey, <http://www.netcraft.com/Survey/>.

- [30] Pitkow, JE and Jones, RK, Supporting the Web: a Distributed Hyperlink Database System, in: The Fifth International World Wide Web Conference, Paris, France, May 6-10 1996.
- [31] Popp, N, Masinter, L, The RealName System: a Human Friendly Naming Scheme, Internet Draft, draft-popp-realname-hfn-00.txt. See also www.RealNames.com.
- [32] RealNames.com, Internet Keyword Subscription Policy, January 2001, http://web.realnames.com/Virtual.asp?page=Eng_Policy_Subscribe_Agreement
- [33] Ross, KW, Hash Routing for Collections of Shared Web Caches, *IEEE Network*, November/December (1997), pp. 37-44.
- [34] Sedlar, E and Clemm, G, Access Control Extensions to WebDAV, Internet Draft, <http://www.webdav.org/acl/protocol/draft-ietf-webdav-acl-01.htm>, April 28 2000.
- [35] Shafer, K, Weibel, S, Jul, E and Fausey, J, Introduction to Persistent Uniform Resource Locators, in: Proceedings of INET96, Montreal, Canada, 24-28 June 1996.
- [36] Sollins, K, Architectural principles of Uniform Resource Name Resolution, RFC 2276, January 1998, <ftp://ftp.isi.edu/in-notes/rfc2276.txt>.
- [37] Sullivan, D, Goodbye Domain Names, Hello Real Names, in: The Search Engine Report, May 2000, <http://www.searchenginewatch.com/sereport/00/05-realnames.html>.
- [38] Sullivan, T, All Things Web, <http://www.pantos.org/atw/35654.html>.
- [39] Sun, SX and Lannom, L, The Handle System: A Persistent Global Name Service – Overview and Syntax, Internet-draft, February 2000, <http://www.ietf.org/internet-drafts/draft-sun-handle-system-04.txt>.
- [40] Thaler, D.G., and Ravishankar, C.V., “Using Name Based Mappings to Increase Hit Rates”, *IEEE/ACM Transactions on Networking*, 6(1), Feb. 1998.
- [41] Valloppillil, V and Ross, KW, Cache Array Routing Protocol v1.0, Internet Draft, draft-vinod-carp-v1-02.txt, <http://www.cs-ipv6.lancs.ac.uk/ipv6/documents/standards/general-comms/internet-drafts/draft-vinod-carp-v1-03.txt>, February 26 1998.